iLMT: Internal Loan Management Tool

Proyecto de fin de carrera

Universidad Carlos III de Madrid

Autor:
Hugo Cárdenas

Supervisoras:
Teresa Onorati
Marijke Thierens

12 de diciembre de 2013

Resumen

Este proyecto presenta una solución para mejorar los procesos logísticos del servicio técnico de hardware de una gran empresa multinacional.

El proyecto se desarrolla en forma de una aplicación web que será utilizada por los empleados del departamento de logística de la empresa. A través de ella podrán gestionar el estado de los componentes de hardware utilizados en el servicio técnico y de los transportes y empleados que toman parte en el mismo.

Gracias a las funcionalidades de la aplicación y a los datos almacenados en ella, los empleados podrán minimizar los tiempos de espera en la toma decisiones y acelerar los procesos de trabajo. Adicionalmente, la herramienta facilitará un análisis de los detalles de manera periódica, con el fin de localizar los cuellos de botella y reducir costes para la empresa.

El diseño de la aplicación se ha llevado a cabo siguiendo los requisitos especificados por la empresa y los casos de uso definidos durante la fase de análisis.

Keywords: web, software, MVC, logística, servicio técnico, componentes de hardware.

Agradecimientos

Me gustaría agradecer a mis tutoras Teresa Onorati y Marijke Thierens el apoyo recibido durante el largo desarrollo de este proyecto.

Helsinki, 12 de diciembre de 2013

Hugo Cárdenas García

Índice general

Índice de figuras

Capítulo 1

Introducción

Este documento presenta como proyecto de fin de carrera una herramienta cuya finalidad es facilitar la gestión diaria en el departamento de logística del servicio técnico informático de una empresa IT a escala nacional.

El departamento se encarga de todos los procesos logísticos relacionados con el servicio técnico de hardware que la empresa ofrece a sus clientes, tanto a particulares como a empresas: desde servidores y sistemas grandes en empresas hasta componentes de ordenadores portátiles de clientes particulares.

La solución propuesta es una aplicación web para uso interno del departamento que, utilizada de manera adicional a las herramientas ya en uso, permita aumentar la eficiencia de los procesos de trabajo y optimizar resultados.

La herramienta permite gestionar toda la información relacionada con los componentes de hardware utilizados en las incidencias del soporte técnico y realizar el seguimiento de los mismos durante su ciclo de vida. El problema a tratar consiste principalmente en el seguimiento de los componentes de hardware siendo de máxima importancia el siguiente requerimiento: todos los componentes deben ser devueltos a los almacenes de la empresa después de su uso en una reparación del servicio técnico.

Tanto los componentes defectuosos o estropeados (después de haber sido reemplazados por los nuevos en la reparación) como los componentes no utilizados en la reparación deben ser devueltos a la empresa y dicha devolución debe quedar registrada. El principal objetivo de este proyecto es contribuir a mejorar y optimizar los procesos que se llevan a cabo para finalizar con éxito las devoluciones de los componentes.

Es cierto que previa a la realización de este proyecto ya existen algunas herramientas que proveen funcionalidad en un ámbito similar, pero con carencias que justifican la creación de este proyecto, que será utilizado de manera combinada con dichas herramientas.

Para llevar a cabo este proyecto se ha realizado inicialmente un análisis del problema a tratar. A través de múltiples reuniones con el cliente se ha realizado una especificación del problema, extrayendo requisitos de usuario y definiendo casos de uso.

Posteriormente, se ha realizado el diseño e implementación de la solución, tomando como base el análisis realizado y clarificando cualquier duda con el cliente en reuniones adicionales.

Al no poseer experiencia previa en este área de conocimiento, las fases de diseño e implementación han estado ligadas, de manera que ambos procesos se han realizado de manera iterativa: inicialmente diseñando la arquitectura de la aplicación a gran escala y a continuación comenzando la implementación, evolucionando ambos procesos en paralelo. Durante el proceso de implementación se han encontrado numerosos obstáculos que han contribuido a reconocer problemas en la arquitectura, que se han ido corrigiendo poco a poco, influyendo estos cambios nuevamente en la implementación.

1.1. Estructura del documento

A continuación se describe la estructura del documento.

El primer capítulo es la presente introducción, en la que se resume el problema y estructuración del documento.

El segundo capítulo es el estado de la cuestión, en el que se describen las contribuciones más relevantes acerca del problema a tratar y el análisis de las tecnologías que se van a utilizar para desarrollar la solución propuesta.

En el tercer capítulo, el planteamiento del problema presenta el problema de una forma general y describe los objetivos principales de este proyecto, que deben ser cumplidos por la solución propuesta.

En el cuarto capítulo, análisis del problema, se detalla el problema y su análisis sistemático, incluyendo requisitos de usuario, requisitos de software y casos de uso, con el fin de tener una especificación a partir de la cual realizar

un diseño de la solución.

El quinto capítulo es el diseño de la solución, en el que se examina en detalle la solución resultante y su proceso de desarrollo, especificando la arquitectura de la aplicación, los detalles relevantes del código y las decisiones tomadas relativas a ambos puntos.

En el sexto capítulo se ofrecen las conclusiones resultantes del desarrollo de este proyecto, incluyendo posibles mejoras y modificaciones.

El séptimo capítulo, gestión del proyecto, recoge las cuestiones relativas a la planificación del trabajo realizado y el tiempo empleado en cada fase del proyecto.

A continuación, el glosario registra los términos del documento que necesitan explicación o aclaración.

Por último, la bibliografía recoge todos los enlaces a obras, páginas web o recursos utilizados para la elaboración de este documento.

Capítulo 2

Estado de la cuestión

En este capítulo se analiza el contexto en el que se desarrolla este proyecto, las herramientas existentes y disponibles en un ámbito de uso similar y las decisiones tomadas acerca de las tecnologías a utilizar.

Actualmente en la empresa hay varias herramientas en uso para gestionar los procesos de logística:

- Existe una herramienta que funciona de manera global para todas las divisiones de la compañía en Europa. Esta herramienta se actualiza diariamente con información del estado y situación de los componentes.
- Al mismo tiempo, el departamento español utiliza varias herramientas propias de manera local.

Por motivos de confidencialidad, no es posible incluir una especificación detallada de las herramientas utilizadas, pero a continuación se describen las funcionalidades que éstas proporcionan, y las carencias que crean la necesidad de desarrollar una nueva aplicación.

Las funcionalidades proporcionadas por las herramientas en uso son las siguientes:

- Listar qué componentes de hardware están pendientes de ser devueltos a la empresa.
- Localizar dónde se encuentran dichos componentes y quién fue la última persona en tratarlos (el ingeniero del servicio técnico ofrecido por la empresa, el transportista, el cliente, etc.)

- Ofrecer procedimientos alternativos para intentar recuperar y procesar aquellos componentes que no se consiguen recuperar siguiendo el procedimiento estándar. Permitir al usuario registrar los posibles desenlaces del problema (componente extraviado, cliente se niega a devolverlo, transportista se niega a devolverlo, etc.)

Las carencias fundamentales de las herramientas existentes son las siguientes:

- La herramienta es actualizada de manera centralizada para Europa únicamente una vez al día.
- La información almacenada relativa a las incidencias y transportes es insuficiente.
- La información editable relativa a las incidencias y componentes es limitada. Muchos de los campos mostrados para una incidencia no se pueden editar.

A continuación se especifican las funcionalidades y características requeridas por la empresa para la nueva aplicación, que posibilitan la mejora del proceso de trabajo actual:

- Se requiere una mayor rapidez en procesar los casos de devoluciones en los que hay problemas. También debe haber más datos disponibles acerca de la situación para poder tomar una mejor solución y reaccionar más rápido.
- Frente a los componentes perdidos, es necesario contar con datos suficientes para poder analizar las causas de los incidentes y encontrar la fase del proceso en la que se encuentran los problemas.
- La empresa requiere realizar la elaboración de informes estadísticos más complejos y útiles, siendo esto posible gracias al almacenamiento de datos más detallados.
- Existe la necesidad de una interfaz sencilla de búsqueda compuesta de numerosos campos adicionales y más eficiente y rápida que las interfaces de búsqueda existentes.
- Es necesario un nuevo proceso de actualización de la base de datos de la aplicación a partir de datos en ficheros (feeds) más rápido que los procesos de actualización para las aplicaciones existentes.

Al no almacenar información suficiente y carecer de una actualización más

frecuente, es imposible mejorar todo el proceso logístico debido a la falta de datos necesarios para ello.

La logística de la empresa en España lleva mucho tiempo funcionando correctamente, pero es en la dificultad de optimizar este proceso donde surge la necesidad para esta herramienta.

Mediante una herramienta que proporcione las funcionalidades requeridas, se puede analizar todo el proceso más detalladamente, identificando los problemas y localizando los cuellos de botella, con la consiguiente mejora de los subprocesos y la optimización de los resultados.

2.1. Tecnologías analizadas

En este capítulo se analiza una selección de tecnologías existentes para desarrollar una aplicación de software, valorando las ventajas e inconvenientes de cada una, seleccionando las que se utilizarán en el desarrollo y argumentando las decisiones tomadas.

2.1.1. Tipo de aplicación

A la hora de elegir el tipo de aplicación a desarrollar, la decisión se plantea básicamente entre dos tipos: aplicación web o aplicación de escritorio.

Para este caso se ha decidido desarrollar una aplicación web, siguiendo una arquitectura cliente-servidor, con una base de datos central y múltiples usuarios que acceden a ella mediante una interfaz.

En las secciones a continuación, se analizarán las tecnologías utilizadas para la implementación de la aplicación, tanto en el lado del servidor, como en el lado del cliente.

2.1.2. Lenguaje de servidor

Como lenguaje de servidor se han considerado las alternativas más comunes: PHP, Python, ASP, Java, Ruby.

Otras aplicaciones previamente implementadas en uso por el departamento de

logística de la empresa están basadas en tecnologías Microsoft, concretamente aplicaciones programadas en ASP que se ejecutan en un servidor IIS. Esto ofrece la ventaja de un entorno ya implementado, reduciendo la necesidad de instalación y configuración del entorno del servidor.

Por otro lado, elegir uno de los lenguajes que siguen una filosofía de código abierto (como PHP o Python) y que tienen una mayor comunidad de seguidores y desarrolladores facilita el aprendizaje y la resolución de los posibles problemas que puedan aparecer.

Figura 2.1: Logo de PHP

Finalmente se ha escogido PHP (logo en figura 2.1) por varias razones:

- Lo he utilizado en anteriores proyectos y por lo tanto cuento con cierta experiencia en su utilización.
- Es de código abierto y cuenta con una gran comunidad de seguidores en internet, siendo uno de los lenguajes más usados para el desarrollo de aplicaciones web [13]. Esto permite que sea más fácil encontrar apoyo de la comunidad en internet, para solucionar los problemas que se encuentren a lo largo del desarrollo. Otra consecuencia de esto es que existe una gran cantidad de módulos y extensiones para proporcionar distintas funcionalidades.
- Es uno de los elementos del entorno LAMP, en el que todos los elementos cuentan con características similares que facilitan su uso en conjunto.

  LAMP es un un término que define un sistema en el que se ejecutan un conjunto determinado de herramientas de código abierto:

  - Linux como sistema operativo.
  - Apache como servidor web.
  - MySql como sistema gestor de base de datos.
  - PHP como lenguaje de programación de servidor.

Se ha valorado también la posibilidad de utilizar alguno de los múltiples frameworks de desarrollo web para PHP, como son:

- Codeigniter: framework creado en 2006 por la empresa EllisLab. Su principal ventaja frente a otros frameworks es su velocidad de ejecución, gracias a una estructura ligera. Por el contrario, ofrece menos funcionalidades que otros frameworks [4].

- Zend Framework: creado en 2006 por la compañía Zend Technologies. Es un framework muy completo, flexible y muy estable. Por otro lado, requiere un gran conocimiento de PHP y programación orientada a objetos, además de cierto trabajo de configuración para adaptarlo a los requerimientos de la aplicación a desarrollar [15].

- Symfony: es un framework creado en 2005 por SensioLabs. Es un sistema muy completo y con cierta complejidad. Entre sus ventajas destacan: buen sistema de *ORM*, generadores de código, facilidad para realizar inyección de dependencias (ver términos en el glosario). Por el contrario, tiene una gran curva de aprendizaje [11].

- Cake PHP: framework creado en 2005 por Cake Software Foundation. Adecuado para pequeños y medianos proyectos, ofrece facilidad de instalación y configuración. La documentación ofrece un nivel de detalle inferior a la de otros frameworks [3].

Finalmente la elección ha sido no utilizar ningún framework por las siguientes razones:

- La no utilización de un framework obliga a comprender todos los detalles del lenguaje y de la estructura de la aplicación, al no existir capas intermedias sobre las que basar el trabajo, más que la proporcionada por el propio lenguaje.

- Mayor probabilidad de cometer errores en el diseño e implementación si se utiliza un framework sin experiencia previa en el desarrollo de aplicaciones web MVC y no se comprende correctamente la estructura y los elementos que lo componen.

- El código desarrollado es mucho más ligero y adaptado a las necesidades del problema. Por un lado, se limita a las funcionalidades que realmente se van a utilizar, dejando aparte las demás funcionalidades del framework. Por otro lado, es necesario realizar un desarrollo extra, debido a carecer de numerosas funcionalidades básicas que los frameworks proporcionan.

2.1.3. Servidor web

En cuanto a la tecnología usada como servidor web, la elección se ha planteado principalmente entre dos opciones: IIS y Apache (logo en figura 2.2).

IIS, por ser el servidor ya instalado en el departamento y utilizado por otras aplicaciones y Apache, por ser un servidor que ya he utilizado y ser el servidor de uso más extendido. También en este caso se cuenta con una gran comunidad de usuarios y soporte en internet.

Figura 2.2: Logo de Apache

Finalmente se ha escogido este último por su mayor compatibilidad con los demás elementos (entorno LAMP) y su facilidad de instalación y configuración.

2.1.4. Base de datos

A lo largo del proyecto se ha producido un cambio de sistema de base de datos. Esto ha sido un hecho interesante para observar cómo se han enfrentado el diseño y la arquitectura de la aplicación a un cambio de este tipo y cuál es la flexibilidad y adaptación que posee para el cambio.

Inicialmente ha sido requisito de la empresa utilizar Microsoft Access como sistema de base de datos, por ser el sistema utilizado y conocido por los demás empleados, quienes utilizarán la base de datos en su trabajo diario y no solo mediante la aplicación.

Posteriormente, al finalizar el período de trabajo en la empresa antes de concluir el proyecto, con el fin de trabajar en él sin tener acceso a Microsoft Access, se tomó la decisión de migrar la base de datos a MySql.

Figura 2.3: Logo de MySql

MySql (logo en figura 2.3) es un sistema de gestión de bases de datos relacional y de código abierto. Es el sistema más ampliamente utilizado en el mundo.

La migración a MySql ofrece las siguientes ventajas:

- Permite la continuación del trabajo en el proyecto en un entorno Linux, sin necesidad de utilizar ningún software de pago.
- Implementación de la base de datos de una manera mucho más robusta y sencilla de modificar (fichero de código sql con la definición completa de la base de datos).
- Dota de mayor apoyo técnico gracias a la enorme comunidad de usuarios en internet.

2.1.5. Lenguaje de cliente

El haber elegido el entorno web como tipo de aplicación condiciona básicamente a utilizar un lenguaje ejecutado por el navegador web en el lado del cliente, como por ejemplo Javascript.

Durante el desarrollo del proyecto, han surgido algunas alternativas minoritarias a Javascript como Dart, lenguaje desarrollado por Google. También existen otros lenguajes que son compilados a Javascript (CoffeeScript, LiveScript, RubyJS, etc.).

Finalmente se ha seleccionado Javascript por su uso mayoritario (que implica mayor facilidad en la búsqueda de recursos en internet) y menor complejidad que un lenguaje intermedio.

Javascript cuenta con múltiples y diferentes librerías, adecuadas para distintos objetivos y variables en complejidad.

Se ha optado por utilizar la librería jQuery (logo en figura 2.4) por su rápida curva de aprendizaje, facilidad de uso y amplia difusión.

Figura 2.4: Logo de JQuery

Las funcionalidades básicas de Javascript que son clave para la aplicación y que jQuery simplifica enormemente gracias a la interfaz que provee son las siguientes:

- Manipulación del DOM HTML.
- Funciones para realizar peticiones AJAX.

Además de jQuery, se utiliza también el framework Bootstrap (logo en figura 2.5), que provee funcionalidad tanto Javascript como CSS, y ha sido creado por Twitter.
Bootstrap se compone de una colección de plantillas de diseño basadas en HTML y CSS, además de funcionalidad extra Javascript. Esto permite desarrollar la interfaz de usuario de una manera mucho más sencilla y rápida, ya que únicamente hay que utilizar ciertos nombres CSS y etiquetas HTML, y el propio framework genera un diseño y estructura totalmente personalizable.

Figura 2.5: Logo de Twitter Bootstrap

Capítulo 3

Planteamiento del problema

El problema que se plantea en este proyecto es la carencia de las herramientas necesarias para almacenar toda la información acerca de los procesos logísticos del servicio técnico que provee la empresa, con el fin de poder extraer datos relevantes y mejorar dichos procesos.

De este problema se deriva la necesidad de desarrollar una herramienta que facilite la gestión de los procesos logísticos diarios del departamento y que almacene al mismo tiempo todos los datos requeridos, que serán de utilidad para elaborar estadísticas y así encontrar los puntos débiles o cuellos de botella de los procesos.

La empresa ofrece un servicio de soporte técnico, tanto a otras empresas como a particulares, para las máquinas y dispositivos manufacturados por ella.

Hay dos tipos de servicio técnico:

- En el primer tipo, el servicio prestado incluye un ingeniero/técnico de la empresa que viaja al lugar indicado a realizar la reparación (generalmente sustituyendo el componente estropeado). Posteriormente, el ingeniero devuelve a la empresa el componente estropeado y los componentes que no se han utilizado en la reparación.
- En el segundo tipo de servicio es el cliente el que se hace cargo de la reparación. El componente necesario se envía al cliente para que arregle el equipo y posteriormente devuelva el componente defectuoso a la empresa.

El almacenamiento de los componentes se realiza en diferentes almacenes, que se engloban en tres grupos:

- Almacén principal de la empresa en Europa localizado en Holanda.
- Almacén principal en España localizado en Guadalajara.
- Múltiples almacenes secundarios en España localizados en puntos clave de la península y Gran Canaria.

El proceso logístico del que se encarga el departamento para el que se realiza la aplicación está compuesto por los siguientes pasos:

1. El cliente contacta con la empresa y comunica su problema. Se abre una incidencia describiendo el problema del cliente.
2. El componente se envía hasta el almacén más cercano al cliente. El proceso por defecto es que desde el almacén central europeo viaje hasta el almacén central español y de éste al almacén secundario más cercano al cliente, aunque es posible que alguno de los pasos se omita si hay existencias del componente requerido en el almacén principal español o en el almacén secundario más cercano.
3. El componente es enviado al cliente, en caso de que no sea necesario servicio técnico, o recogida por un técnico, quien realiza la visita al cliente y efectúa la reparación.
4. Todos los componentes sobrantes de la reparación (componente defectuoso sustituido por el nuevo y posibles componentes no utilizados en la reparación) son devueltos al almacén. Los clientes los envían mediante mensajería, o mediante el mismo técnico, quien los devuelve personalmente al almacén donde hizo la recogida previamente.
5. Los componentes sin utilizar y los componentes defectuosos con posibilidad de ser reparados son enviados al almacén principal español, desde el que posteriormente son enviados al almacén principal europeo.

Como consecuencia de este proceso, la información principal del sistema gira en torno a los componentes y a su ciclo de vida. El ciclo de vida de un componente comienza cuando se abre una incidencia en la que el componente es requerido. Es posible que haya componentes que no se utilicen en la reparación.

El ciclo termina en la mayor parte de los casos cuando el componente sustituido

(defectuoso) que se ha reemplazado por el nuevo, o el componente nuevo (si no se ha utilizado en la reparación) finaliza su retorno al almacén central europeo, después de haber sido devuelto por el ingeniero o el cliente a la dirección de devolución correspondiente.

Esta dirección puede ser un almacén regional en algunos casos, o el almacén central de la península, según corresponda.

Los tres tipos de almacenes se clasifican mediante los términos internos *FSL* (almacén regional en algún punto de la península), *CHUB* (único almacén central español) y *CRWB* (único almacén central europeo en Holanda).

Hay una excepción para el fin del ciclo de vida de los componentes. Hay ciertos componentes que por diversos motivos no son devueltos a ningún almacén. Puede ser debido a la negativa del cliente o el ingeniero a devolverlos, pérdida, o cualquier otro motivo.

En estos casos, los empleados del departamento de logística intentan localizar los componentes y facilitar su recuperación, mediante contacto con el cliente, ingeniero o servicio de transportes. En caso de transcurrir cierto tiempo sin éxito, los empleados pueden dar por perdido el componente y realizar una petición de ajuste. Ajuste es el término otorgado en el ámbito del proceso logístico a la acción de dar por perdido un componente no devuelto después de una reparación, después de un período de tiempo tratando de recuperarlo sin éxito.

Los empleados pueden realizar peticiones de ajuste para los componentes, pero sólo aquellos empleados cuyo usuario sea de tipo administrador pueden aprobar o rechazar los ajustes. De esta manera, sólo los empleados que poseen este poder de decisión pueden aprobar si se da por perdido un componente. Esta decisión será tomada dependiendo de las condiciones de cada componente concreto.

A partir del proceso descrito previamente se definen los objetivos del problema.

3.1. Objetivos

Los objetivos de este proyecto son los siguientes:

- Proporcionar una herramienta con una interfaz gráfica que pueda ser utilizada por el personal del departamento de logística de la empresa.

- La herramienta debe permitir a los usuarios visualizar y actualizar la información necesaria relativa a los procesos de trabajo del departamento, principalmente:
  - Los datos y el estado de los componentes a utilizar o utilizados en las incidencias del soporte técnico.
  - Datos y estado de los transportes utilizados para el traslado de los componentes, tanto antes como después de la reparación.
- La herramienta debe ser interoperable, es decir, debe permitir el uso a múltiples usuarios de manera simultánea, compartiendo la misma fuente de datos.
- La herramienta debe ser independiente de la plataforma, es decir, los usuarios deben poder utilizarla en diferentes sistemas operativos de ordenador. No es requisito el soporte para otros dispositivos, como móvil o tablet.
- La herramienta debe contar con una base de datos que almacene toda la información necesaria acerca de los componentes y las incidencias relacionadas. Esta base de datos debe cumplir una especificación definida por la empresa, para almacenar todos los datos necesarios para elaborar posteriores informes y estadísticas.
- Se debe proporcionar un mecanismo que permita la actualización de la base de datos con la situación actual de todos los componentes, utilizando como fuente de datos múltiples feeds en ficheros que serán actualizados regularmente.
  El sistema permitirá al usuario actualizar la base de datos con la frecuencia deseada, siempre que los feeds hayan sido actualizados, siendo ésto último no dependiente del sistema. El único límite del sistema a la frecuencia de actualización será el tiempo de ejecución de la misma, ya que no pueden ejecutarse múltiples actualizaciones concurrentes a partir del mismo feed.

Capítulo 4

Análisis del problema

El proceso de análisis de un problema de desarrollo de software comprende las siguientes fases:

- Definir el problema.
- Extraer los requisitos de usuario y de software.
- Analizar los casos de uso.
- Identificar subsistemas de análisis.

4.1. Definición del problema

En esta sección se describe el problema indentificado en el capítulo anterior, delimitando su alcance, definiendo el entorno tecnológico e identificando los usuarios participantes en el desarrollo de la aplicación y los usuarios que finalmente la utilizarán.

4.1.1. Funcionalidades del sistema

El sistema requerido por el cliente debe permitir la gestión de los componentes utilizados y el transporte de los mismos en el servicio de soporte técnico de la empresa.

Debe contar con las siguientes funcionalidades:

- El sistema debe permitir realizar una búsqueda de componentes y visualizar y modificar los detalles y estado de un componente seleccionado.
- El sistema debe permitir realizar una búsqueda de transportes y visualizar y modificar los detalles y estado de un transporte seleccionado.
- El sistema debe permitir crear, editar y eliminar la información acerca de los ingenieros que trabajan en las incidencias del servicio de soporte técnico.
- El sistema debe permitir crear, editar y eliminar usuarios del propio sistema.

Tendrá las siguientes restricciones de carácter técnico y operativo:

- El sistema debe ser accesible a través de una red desde los navegadores Firefox, Internet Explorer, Google Chrome, Opera y Safari.
- El sistema deberá ser seguro y mantenible.
- La organización cliente no dispone de un Plan de Sistemas de Información, por lo que no será necesario tener en cuenta ninguna arquitectura de información adicional para analizar el alcance del sistema.

A continuación se expone el modelo conceptual del sistema, que muestra las entidades contenidas en la aplicación y las relaciones existentes entre ellas:

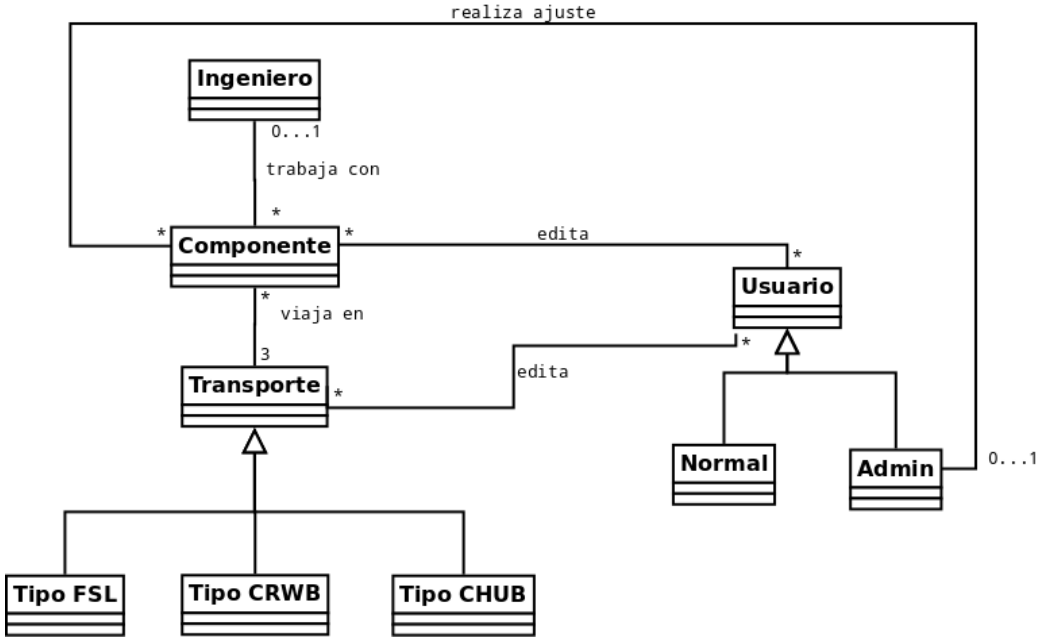

Figura 4.1: Modelo conceptual del sistema

El modelo conceptual recoge de forma básica los principales elementos del problema, las relaciones entre ellos y facilita la comprensión del mismo.

Los dos elementos básicos del problema son los *componentes* utilizados en las reparaciones del servicio técnico y los *transportes* en los que dichos *componentes* viajan hasta su destino. Cada *componente* puede estar asociado con hasta tres *transportes*, uno de cada tipo.
Los tipos de *transporte* están asociados a sus destinos:

- *FSL*: *Transportes* cuyo destino es un almacén regional en la península.
- *CHUB*: *Transportes* cuyo destino es el almacén central español.
- *CRWB*: *Transportes* cuyo destino es el almacén central europeo.

Los *ingenieros* son los empleados del servicio técnico de la empresa que realizan la reparación en la localización requerida por el cliente.
Un componente estará asociado al *ingeniero* que realiza la reparación, en caso de que el servicio prestado para la incidencia concreta requiera la asistencia del *ingeniero*. En caso de que el servicio técnico se limite a enviar el *componente* al cliente para que éste mismo realice la reparación, no hay ningún *ingeniero* asociado al *componente*.

Los *usuarios* son los empleados del departamento de logística que hacen uso de la aplicación resultado de este proyecto. Éstos pueden editar la información de los *componentes* y los *transportes*. Además pueden realizar peticiones de ajuste de los *componentes* (ver glosario), que son validadas por los *usuarios* de tipo *administrador*.

La información de un *componente* está constituida por un gran número de campos, que se estructuran en varias entidades para facilitar el tratamiento de los datos:

- Información principal del *componente*: engloba todos los datos básicos que deben ser utilizados para realizar búsquedas y listados de *componentes*, y para la identificación de los mismos.
- Comment: recoge los comentarios registrados por los *usuarios* en *componentes* específicos.
- Operation: recoge información acerca de los costes que suponen para la empresa el empleo de los *componentes* en las reparaciones.
- Rev_receipt: recoge información referente al almacén que recibe los *componentes* devueltos después de una reparación.
- Blind_receipt: recoge información extra en los casos en los que hay algún problema con el registro e identificación de los *componentes* devueltos al almacén *CRWB* (almacén central europeo).
- Handling: recoge información extra en los casos en los que hay algún problema con el registro e identificación de los *componentes* devueltos a un almacén *FSL* (almacén regional) o a un almacén *CHUB* (almacén central español).

Para más detalles acerca de estos campos, ver la definición del modelo de datos en el apartado 5.4.1.

4.1.2. Estándares y normas

La realización de este proyecto conlleva el cumplimiento de los estándares establecidos a tener en cuenta para elaborar el diseño del sistema:

- UML 2.0: Estándar que define el lenguaje UML (Unified Modelling

Language) para el modelado de sistemas de software.

- SQL:2003: Estándar que define el lenguaje SQL para bases de datos relacionales.
- HTML 4.01: Estándar que define el lenguaje de marcado HTML para el desarrollo de documentos web.

4.1.3. Usuarios

El cliente participará en los procesos de validación y aceptación del sistema. Los usuarios finales se clasifican en dos roles, que se describen a continuación:

Rol	Descripción
Básico	Es el usuario estándar de la aplicación. En este rol se clasifican la mayoría de empleados del departamento. Tendrá acceso a las principales funcionalidades de la aplicación: búsqueda y edición de componentes, búsqueda y edición de transportes y actualización de la base de datos.
Administrador	Este usuario tiene acceso a todas las funcionalidades del usuario básico y además posee privilegios de administración del sistema. Las funcionalidades de administración son las siguientes: realizar ajustes de componentes, edición de ingenieros y edición de usuarios del sistema.

Tabla 4.1: Tipos de usuarios

4.2. Definición de requisitos

A continuación se definen los requisitos extraídos durante el análisis del problema.

4.2.1. Requisitos de usuario

Los requisitos de usuario describen el sistema desde el punto de vista del cliente y del usuario final. Definen los requerimientos del sistema a alto nivel de forma que sean comprensibles por los usuarios que no posean un conocimiento técnico detallado.

Cada requisito de usuario queda especificado por los siguientes atributos:

- Nombre del requisito.
- Descripción del requisito.
- Tipo de usuario al que se aplica la funcionalidad descrita.
- Prioridad del requisito con respecto al proceso de desarrollo.
- Necesidad del requisito en la aplicación.
- Estabilidad del requisito. Este atributo define si un requisito tiene probabilidades de cambiar su especificación en un futuro.
- Prerrequisito para que el requisito pueda cumplirse.
- Fuente u origen de la creación del requisito.

Lista de requisitos de usuario:

Identificador: RU001	
Nombre	Buscador de componentes
Descripción	La aplicación debe contener un buscador de componentes que permita buscar mediante una lista especificada de campos (ver sección 5.4.2).
Usuario	Básico
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU002	
Nombre	Editor de componentes
Descripción	La aplicación debe contener un editor de componentes, en el que se muestren todos los detalles de un componente y que permita editar aquellos campos definidos como editables (ver sección 5.4.2).
Usuario	Básico
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU003	
Nombre	Buscador de transportes
Descripción	La aplicación debe contener un buscador de transportes que permita buscar mediante una lista especificada de campos (ver sección 5.4.2).
Usuario	Básico
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU004	
Nombre	Editor de transportes
Descripción	La aplicación debe contener un editor de transportes, en el que se muestren todos los detalles de un transporte y que permita editar aquellos campos definidos como editables (ver sección 5.4.2).
Usuario	Básico
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU005	
Nombre	Actualización de la base de datos
Descripción	La aplicación debe contar con un sistema mediante el que actualizar la base de datos utilizando como entrada una serie de feeds en formato csv.
Usuario	Básico
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Es necesario un diseño previo de la base de datos.
Fuente	Cliente

Identificador: RU006	
Nombre	Ajuste de componentes
Descripción	La aplicación debe contar con un sistema mediante el que realizar el ajuste de componentes
Usuario	Básico, Administrador
Prioridad	Media
Necesidad	Alta
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU007	
Nombre	Niveles de privilegios
Descripción	La aplicación debe contar con un sistema para determinar el nivel de privilegios de los diferentes usuarios y permitir el acceso a diferentes funcionalidades según corresponda.
Usuario	Administrador
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU008	
Nombre	Gestión de usuarios
Descripción	La aplicación debe permitir la gestión de usuarios de la propia aplicación, ofreciendo la posibilidad de crear, editar y eliminar usuarios.
Usuario	Administrador
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

Identificador: RU009	
Nombre	Gestión de ingenieros
Descripción	La aplicación debe permitir la gestión de la información acerca de los ingenieros de la empresa, ofreciendo la posibilidad de crear, editar y eliminar ingenieros.
Usuario	Administrador
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Prerrequisito	Ninguno
Fuente	Cliente

4.2.2. Requisitos de software

Los requisitos de software describen el sistema desde el punto de vista de los ingenieros de software. Definen los requerimientos del sistema con detalle con el propósito de proporcionar a los programadores información suficiente para implementar el sistema. Se dividen en requisitos funcionales y no funcionales.

Cada requisito de software queda especificado por los siguientes atributos:

- Nombre del requisito.
- Descripción del requisito.
- Prioridad del requisito con respecto al proceso de desarrollo.
- Necesidad del requisito en la aplicación.
- Estabilidad del requisito. Este atributo define si un requisito tiene probabilidades de cambiar su especificación en un futuro.
- Origen del requisito (para los requisitos funcionales, será una referencia a un requisito de usuario).

Requisitos funcionales

Los requisitos de software funcionales definen las funcionalidades que el sistema debe implementar para satisfacer las necesidades del usuario. Son derivados de los requisitos de usuario.

Identificador: RSF001	
Nombre	Buscador de componentes
Descripción	El buscador de componentes de la aplicación debe permitir realizar una búsqueda basada en uno o varios de los campos especificados en la lista correspondiente (ver sección 5.4.2).
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU001

Identificador: RSF002	
Nombre	Resultado de búsqueda de componentes
Descripción	El resultado de la búsqueda de componentes debe ser la lista de los componentes que cumplen el criterio de búsqueda. Además, cada resultado de búsqueda debe contener un enlace que conduzca a la página de detalles/edición del componente.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU001

Identificador: RSF003	
Nombre	Edición de componente
Descripción	El editor de componentes debe mostrar todos los detalles de un componente. Debe permitir al usuario editar los campos definidos como editables (ver sección 5.4.2) y guardar los cambios.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU002

Identificador: RSF004	
Nombre	Actualización de estado de componente
Descripción	Siempre que se modifica un componente y se guardan los cambios, se debe aplicar un algoritmo que compruebe los detalles del componente y actualice el estado del mismo en caso de ser necesario.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU002

Identificador: RSF005	
Nombre	Buscador de transportes
Descripción	El buscador de transportes de la aplicación debe permitir realizar una búsqueda basada en uno o varios de los campos especificados (ver sección 5.4.2).
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU003

Identificador: RSF006	
Nombre	Resultado de búsqueda de transportes
Descripción	El resultado de la búsqueda de transportes debe ser la lista de las transportes que cumplen el criterio de búsqueda. Además, cada resultado de búsqueda debe contener un enlace que conduzca a la página de detalles/edición del transporte.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU003

Identificador: RSF007	
Nombre	Edición de transporte
Descripción	El editor de transportes debe mostrar todos los detalles de un transporte. Debe permitir al usuario editar los campos definidos como editables (ver sección 5.4.2) y guardar los cambios.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU004

Identificador: RSF008	
Nombre	Interfaz para actualizar la base de datos
Descripción	La aplicación debe ofrecer una interfaz mediante la cual iniciar las actualizaciones de la base de datos.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU005

Identificador: RSF009	
Nombre	Sistema de actualización de la base de datos
Descripción	La aplicación debe contar con un sistema para actualizar la base de datos, tomando como entrada un conjunto de feeds que contienen la información en un formato específico.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU005

Identificador: RSF010	
Nombre	Lista de componentes para realizar ajuste
Descripción	La aplicación debe ofrecer una interfaz en la que se muestre la lista de los componentes pendientes de ajuste (ver término en el glosario).
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU001

Identificador: RSF011	
Nombre	Ajuste de componentes
Descripción	La aplicación debe permitir seleccionar una lista de componentes y aplicar un ajuste determinado. Sólo los usuarios de tipo Administrador deben tener acceso a esta funcionalidad.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU006

Identificador: RSF012	
Nombre	Niveles de privilegios definidos en el usuario
Descripción	La base de datos debe almacenar los datos de los usuarios de la aplicación. Esta información debe incluir el nivel de privilegios que posee el usuario. Este atributo define las funcionalidades de la aplicación a la que el usuario tiene acceso.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU007

Identificador: RSF013	
Nombre	Niveles de privilegios definidos en las funcionalidades
Descripción	Las diferentes funcionalidades de la aplicación deben definir el nivel de privilegios que requieren para que un usuario pueda tener acceso a ellas. Si un usuario no posee el nivel necesario para usar una funcionalidad determinada, tendrá restringido su uso.
Prioridad	Alta
Necesidad	Alta
Estabilidad	Si
Origen	RU007

Identificador: RSF014	
Nombre	Lista de usuarios
Descripción	La aplicación debe mostrar la lista de los usuarios existentes en el sistema, ofreciendo una interfaz para realizar las operaciones correspondientes (editar/eliminar) sobre cada uno.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU008

Identificador: RSF015	
Nombre	Creación y edición de usuario
Descripción	La aplicación debe permitir la creación de nuevos usuarios y edición de usuarios existentes. La aplicación debe realizar una validación de los datos del usuario y a continuación, mostrar un mensaje satisfactorio al usuario en caso de completarse la operación con éxito o mostrar un mensaje de error en caso de ser los datos incorrectos.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU008

Identificador: RSF016	
Nombre	Eliminación de usuario
Descripción	La aplicación debe permitir la eliminación de los usuarios existentes en el sistema.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU008

Identificador: RSF014	
Nombre	Lista de ingenieros
Descripción	La aplicación debe mostrar la lista de los ingenieros existentes en el sistema, ofreciendo una interfaz para realizar las operaciones correspondientes (editar/eliminar) sobre cada uno.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU009

Identificador: RSF015	
Nombre	Creación y edición de ingeniero
Descripción	La aplicación debe permitir la creación de nuevos ingenieros y edición de ingenieros existentes. La aplicación debe realizar una validación de los datos del ingeniero y a continuación, mostrar un mensaje satisfactorio al usuario en caso de completarse la operación con éxito o mostrar un mensaje de error en caso de ser los datos incorrectos.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU009

Identificador: RSF016	
Nombre	Eliminación de ingeniero
Descripción	La aplicación debe permitir la eliminación de los ingenieros existentes en el sistema.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	RU009

Requisitos no funcionales

Los requisitos de software no funcionales definen requerimientos del sistema no relacionados directamente con funcionalidades requeridas por el usuario.

Identificador: RSNF001	
Nombre	Compatibilidad con navegadores web
Descripción	La aplicación debe ser compatible con los siguientes navegadores web: ▪ Mozilla Firefox, versión 3.6 en adelante. ▪ Google Chrome, versión 7.0 en adelante.
Prioridad	Media
Necesidad	Media
Estabilidad	Si
Origen	Cliente

4.3. Especificación de los casos de uso

El siguiente diagrama muestra la interacción del sistema con los usuarios:

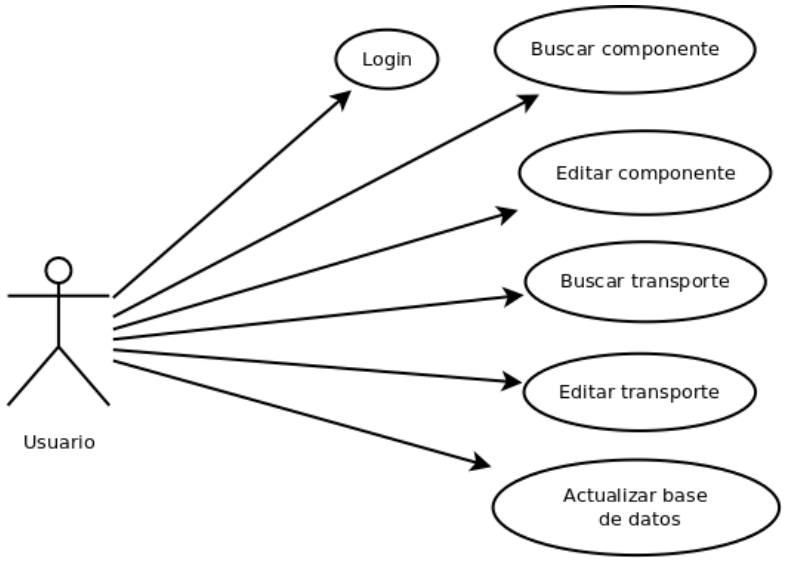

Figura 4.2: Casos de uso: usuario

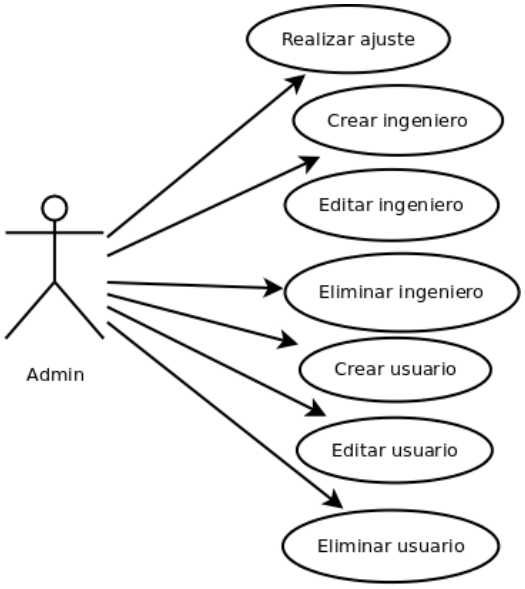

Figura 4.3: Casos de uso: administrador

A continuación se especifican los casos de uso:

Identificador: CU001	
Nombre	Login
Actores	Usuario básico
Objetivo	Realizar el login del usuario en el sistema
Precondiciones	El usuario debe estar registrado en el sistema
Escenario principal	El usuario se encuentra en la pantalla de login. Desde allí: 1. Introduce su nombre en el campo *nombre*. 2. Introduce su contraseña en el campo *contraseña*. 3. Pulsa el botón de login. 4. Se realiza la validación de los datos introducidos. 5. Si los datos son correctos, el sistema muestra al usuario la página principal de la aplicación.
Escenario alternativo	5. Si los datos son incorrectos, el sistema avisa de ello al usuario y le invita a que los introduzca de nuevo.
Postcondiciones	El usuario está logueado en el sistema. Puede utilizar las funcionalidades del sistema acorde con sus privilegios.

Identificador: CU002	
Nombre	Buscar componente
Actores	Usuario básico
Objetivo	Realizar una búsqueda de componentes mediante una lista de criterios de búsqueda
Precondiciones	El usuario debe estar logueado en el sistema
Escenario principal	El usuario se encuentra en la pantalla de búsqueda de componentes. Desde allí: 1. Introduce los criterios de búsqueda en el formulario. 2. Pulsa el botón de búsqueda. 3. Los resultados de la búsqueda son cargados en la página. 4. El usuario puede navegar por los resultados. Cada resultado ofrece un vínculo mediante el que navegar a la página de detalles del componente.
Escenario alternativo	3. No existe ningún resultado. Se muestra un mensaje al usuario con dicha información.
Postcondiciones	Los resultados de la búsqueda están cargados en la página.

Identificador: CU003	
Nombre	Editar componente
Actores	Usuario básico
Objetivo	Visualizar y editar los detalles de un componente
Precondiciones	El usuario debe estar logueado en el sistema
Escenario principal	El usuario se encuentra en la pantalla de detalles de un componente. A continuación: 1. El usuario navega por las secciones de detalles de un componente, hasta encontrar la información que desea modificar. 2. Pulsa el botón correspondiente para editar los campos modificables de dicha sección. 3. Introduce los nuevos datos. 4. Pulsa el botón correspondiente para guardar los cambios.
Escenario alternativo	
Postcondiciones	Los cambios se han hecho permanentes en la aplicación y se muestran en la página de detalles del componente

Identificador: CU004	
Nombre	Buscar transporte
Actores	Usuario básico
Objetivo	Realizar una búsqueda de transportes mediante una lista de criterios de búsqueda
Precondiciones	El usuario debe estar logueado en el sistema
Escenario principal	El usuario se encuentra en la pantalla de búsqueda de transportes. Desde allí: 1. Introduce los criterios de búsqueda en el formulario. 2. Pulsa el botón de búsqueda. 3. Los resultados de la búsqueda son cargados en la página. 4. El usuario puede navegar por los resultados. Cada resultado ofrece un vínculo mediante el que navegar a la página de detalles del transporte.
Escenario alternativo	3. No existe ningún resultado. Se muestra un mensaje al usuario con dicha información.
Postcondiciones	Los resultados de la búsqueda están cargados en la página.

Identificador: CU005	
Nombre	Editar transporte
Actores	Usuario básico
Objetivo	Visualizar y editar los detalles de un transporte
Precondiciones	El usuario debe estar logueado en el sistema
Escenario principal	El usuario se encuentra en la pantalla de detalles de un transporte. A continuación: 1. Pulsa el botón correspondiente para editar los campos modificables del transporte. 2. Introduce los nuevos datos. 3. Pulsa el botón correspondiente para guardar los cambios.
Escenario alternativo	
Postcondiciones	Los cambios se han hecho permanentes en la aplicación y se muestran en la página de detalles del transporte

Identificador: CU006	
Nombre	Actualizar base de datos
Actores	Usuario básico
Objetivo	Actualizar la base de datos a partir de feeds externos
Precondiciones	▪ El usuario debe estar logueado en el sistema. ▪ Los feeds necesarios deben encontrarse en la carpeta correspondiente de la aplicación.
Escenario principal	El usuario se encuentra en la sección de actualización de la base de datos. A continuación: 1. Pulsa uno de los botones para actualizar la base de datos a partir del feed correspondiente (existe un botón por cada feed). 2. Se muestra un mensaje al usuario de espera mientras se realiza la operación. 3. Cuando la actualización finaliza, se muestra un mensaje al usuario conteniendo información acerca del resultado.
Escenario alternativo	
Postcondiciones	La base de datos se ha actualizado con los datos provenientes del feed seleccionado.

Identificador: CU007	
Nombre	Realizar ajuste
Actores	Administrador
Objetivo	Realizar ajuste de un componente
Precondiciones	▪ El usuario debe estar logueado en el sistema y ser de tipo admin. ▪ Debe existir al menos un componente con una petición de ajuste pendiente de ser aprobada.
Escenario principal	El usuario se encuentra en la sección de ajustes. En ella se mostrará una lista de los componentes pendientes de ajuste. A continuación: 1. Selecciona los componentes sobre los que realizar una acción mediante los checkboxes. 2. Pulsa el botón *aprobar*. 3. Los componentes seleccionados actualizan su información de ajuste, especificando que el ajuste ha sido aprobado. La lista de componentes pendientes de ser ajustados se actualiza, desapareciendo de ella los componentes previamente seleccionados sobre los que se ha realizado la acción.
Escenario alternativo	2. Pulsa el botón *rechazar*. 3. Los componentes seleccionados actualizan su información de ajuste, especificando que el ajuste ha sido rechazado. La lista de componentes pendientes de ser ajustados se actualiza, desapareciendo de ella los componentes previamente seleccionados sobre los que se ha realizado la acción.
Postcondiciones	Los componentes sobre los que se ha realizado la acción han actualizado su información en la base de datos, conteniendo la información correspondiente sobre el ajuste aprobado o rechazado, según la acción realizada por el usuario.

Identificador: CU008	
Nombre	Crear ingeniero
Actores	Administrador
Objetivo	Crear un ingeniero
Precondiciones	El usuario debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario se encuentra en la sección ingenieros, donde se muestra la lista de los ingenieros en el sistema. A continuación: 1. Pulsa el botón *crear ingeniero*. 2. Rellena los campos de creación del ingeniero. 3. Pulsa el botón *crear*. 4. Si el contenido de los campos es correcto, el ingeniero se crea en la base de datos, mostrando al usuario un mensaje satisfactorio y actualizando la lista de ingenieros.
Escenario alternativo	4. Si el contenido de alguno de los campos es incorrecto o incompleto, se advierte de ello al usuario para que corrija el error.
Postcondiciones	La base de datos se encuentra actualizada con la información del nuevo ingeniero.

Identificador: CU009	
Nombre	Editar ingeniero
Actores	Administrador
Objetivo	Editar los detalles de un ingeniero
Precondiciones	El usuario debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario se encuentra en la sección ingenieros, donde se muestra la lista de los ingenieros en el sistema. A continuación: 1. Pulsa el botón *editar* correspondiente al ingeniero que se desea editar. 2. Edita los campos deseados del ingeniero. 3. Pulsa el botón *actualizar*. 4. Si el contenido de los campos es correcto, los detalles del ingeniero se actualizan en la base de datos, mostrando al usuario un mensaje satisfactorio y actualizando la lista de ingenieros.
Escenario alternativo	4. Si el contenido de alguno de los campos es incorrecto o incompleto, se advierte de ello al usuario para que corrija el error.
Postcondiciones	La base de datos se encuentra actualizada con la nueva información del ingeniero.

Identificador: CU010	
Nombre	Eliminar ingeniero
Actores	Administrador
Objetivo	Eliminar la información de un ingeniero
Precondiciones	El usuario debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario se encuentra en la sección ingenieros, donde se muestra la lista de los ingenieros en el sistema. A continuación: 1. Pulsa el botón *eliminar* correspondiente al ingeniero que se desea eliminar. 2. Se muestra un diálogo de confirmación al usuario. 3. El usuario confirma que desea eliminar el ingeniero. 4. La información del ingeniero es eliminada de la base de datos y se actualiza la lista de ingenieros.
Escenario alternativo	3. Si en el diálogo de confirmación el usuario cancela la operación, no se produce ningún cambio.
Postcondiciones	La base de datos se encuentra actualizada, sin la información del ingeniero eliminado.

Identificador: CU008	
Nombre	Crear usuario
Actores	Administrador
Objetivo	Crear un usuario
Precondiciones	El usuario actual debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario actual se encuentra en la sección usuarios, donde se muestra la lista de los usuarios en el sistema. A continuación: 1. Pulsa el botón *crear usuario*. 2. Rellena los campos de creación del usuario. 3. Pulsa el botón *crear*. 4. Si el contenido de los campos es correcto, el usuario se crea en la base de datos, mostrando al usuario actual un mensaje satisfactorio y actualizando la lista de usuarios.
Escenario alternativo	4. Si el contenido de alguno de los campos es incorrecto o incompleto, se advierte de ello al usuario para que corrija el error.
Postcondiciones	La base de datos se encuentra actualizada con la información del nuevo usuario.

Identificador: CU009	
Nombre	Editar usuario
Actores	Administrador
Objetivo	Editar los detalles de un usuario
Precondiciones	El usuario actual debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario actual se encuentra en la sección usuarios, donde se muestra la lista de los usuarios en el sistema. A continuación: 1. Pulsa el botón *editar* correspondiente al usuario que se desea editar. 2. Edita los campos deseados del usuario. 3. Pulsa el botón *actualizar*. 4. Si el contenido de los campos es correcto, los detalles del usuario se actualizan en la base de datos, mostrando al usuario actual un mensaje satisfactorio y actualizando la lista de usuarios.
Escenario alternativo	4. Si el contenido de alguno de los campos es incorrecto o incompleto, se advierte de ello al usuario para que corrija el error.
Postcondiciones	La base de datos se encuentra actualizada con la nueva información del usuario.

Identificador: CU010	
Nombre	Eliminar usuario
Actores	Administrador
Objetivo	Eliminar la información de un usuario
Precondiciones	El usuario actual debe estar logueado en el sistema y ser de tipo admin.
Escenario principal	El usuario actual se encuentra en la sección usuarios, donde se muestra la lista de los usuarios en el sistema. A continuación: 1. Pulsa el botón *eliminar* correspondiente al usuario que se desea eliminar. 2. Se muestra un diálogo de confirmación al usuario actual. 3. El usuario actual confirma que desea eliminar el usuario. 4. La información del usuario es eliminada de la base de datos y se actualiza la lista de usuarios.
Escenario alternativo	3. Si en el diálogo de confirmación el usuario cancela la operación, no se produce ningún cambio.
Postcondiciones	La base de datos se encuentra actualizada, sin la información del usuario eliminado.

4.4. Subsistemas de análisis

En esta sección se detalla la descomposición del sistema en subsistemas, que facilitará el análisis del sistema completo.

- El subsistema *componentes* ofrecerá la funcionalidad necesaria para realizar búsquedas de componentes basadas en una serie de campos específicos. Además, permitirá visualizar y editar los detalles de un componente concreto.
- El subsistema *transportes* ofrecerá la funcionalidad necesaria para realizar búsquedas de transportes basadas en una serie de campos específicos. Además, permitirá visualizar y editar los detalles de un transporte concreto.
- El subsistema *actualización de base de datos* ofrecerá la funcionalidad necesaria para actualizar la base de datos del sistema a partir de ciertos ficheros específicos (feeds) que contendrán los datos que deben ser insertados y/o actualizados en la base de datos.
- El subsistema *ajustes* ofrecerá la funcionalidad necesaria para procesar las peticiones de ajuste de los componentes (ver definición en el glosario), realizadas por los usuarios del sistema. Permitirá visualizar todas las peticiones de ajuste y realizar las acciones correspondientes sobre las mismas.
- El subsistema *ingenieros* ofrecerá la funcionalidad necesaria para gestionar (crear, editar y eliminar) la información de los ingenieros del sistema (ver definición en el glosario).
- El subsistema *usuarios* ofrecerá la funcionalidad necesaria para gestionar (crear, editar y eliminar) la información de los usuarios del sistema.

Capítulo 5

Diseño de la solución

El objetivo del diseño del sistema es definir la arquitectura de la aplicación y el entorno tecnológico que le va a dar soporte. Se realiza también la especificación detallada de los componentes que forman parte del sistema.

El diseño del sistema se ha dividido en las siguientes fases:

- Definición de la arquitectura del sistema: en esta fase se define la arquitectura general del sistema, especificando la estructura de la aplicación a gran escala y la interacción entre los principales elementos, sin entrar en detalles específicos.
- Diseño de las clases que compondrán el sistema: esta fase recoge la especificación de las clases que forman la aplicación, explicando con detalle el contenido de las clases y sus interacciones.
- Diseño de la interfaz de usuario: en esta fase se muestra el diseño de las interfaces de usuario que ofrecerá la aplicación.
- Diseño del modelo físico de datos: esta fase recoge la especificación de los datos que trata el sistema y su almacenamiento y estructura en la base de datos.
- Especificación del proceso de migración y carga inicial de datos: en esta fase se describen los procesos necesarios para poner el sistema en funcionamiento por primera vez, y realizar migraciones de aplicaciones previas en uso a la nueva aplicación, en caso de ser necesario.
- Aprobación del sistema: esta fase recoge el proceso de validación del

sistema y su integración en el entorno de trabajo requerido.

5.1. Arquitectura del sistema

En esta sección se describe la arquitectura general del sistema determinando las distintas particiones físicas del mismo y especificando la infraestructura tecnológica necesaria para dar soporte al sistema.

La arquitectura de la aplicación web se ha diseñado siguiendo una estructura de tipo MVC (Modelo-Vista-Controlador).

Los componentes de la arquitectura son los siguientes:

- Modelo: Es la capa mediante la que se trata la información del sistema. Permite tratar los datos, gestionando todas las operaciones como consultas, modificaciones o borrado de datos.
- Controlador: Es la capa que recibe las peticiones del usuario y las procesa. Accede al modelo cuando sea necesario para tratar los datos y los procesa. Por último, transfiere los datos a la vista correspondiente para responder a la acción del usuario.
- Vista: Es la capa que presenta los datos al usuario. Se encarga de aplicar el formato correspondiente a los datos resultantes de la acción del controlador.

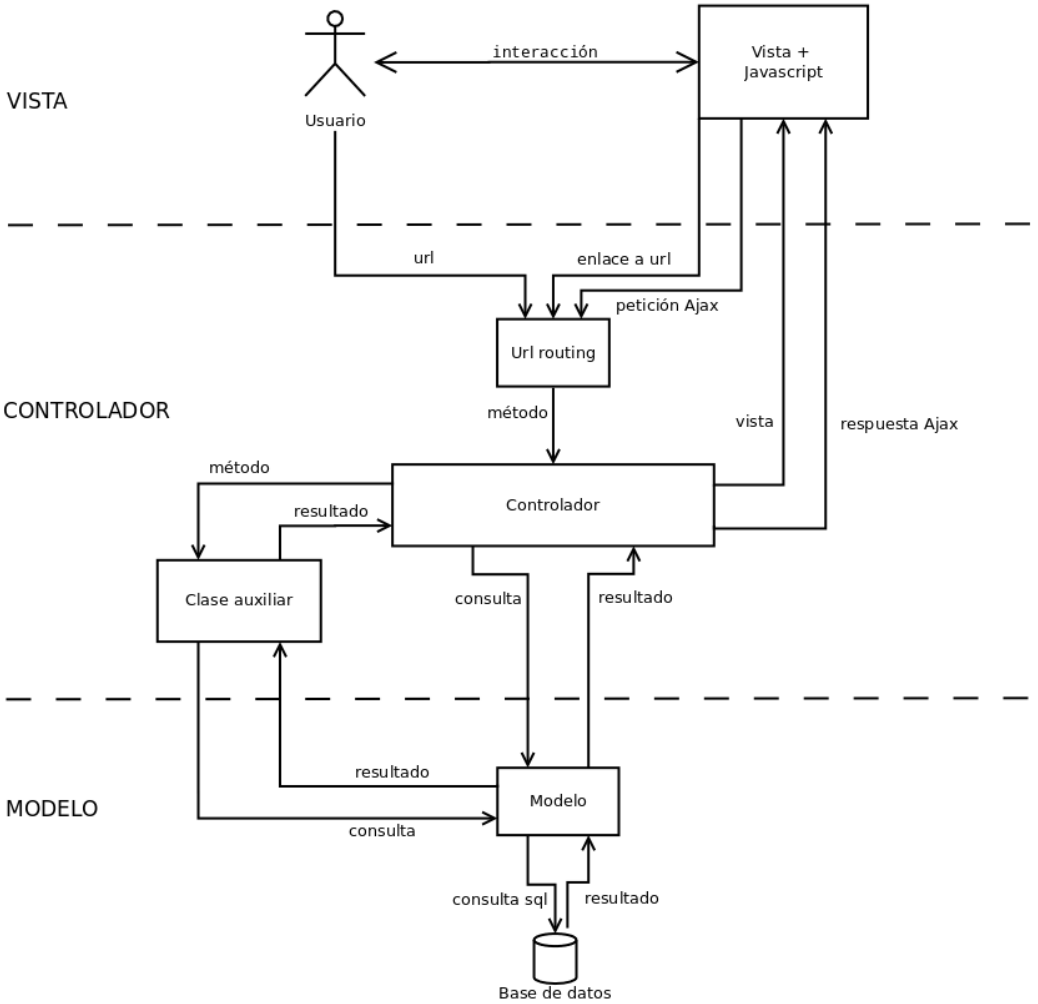

Figura 5.1: Diagrama de arquitectura

En la figura 5.1 se puede ver con algo más de detalle la estructura real de la aplicación. El funcionamiento básico de la arquitecura MVC tiene el siguiente flujo:

1. El usuario hace una petición a través de una url.

2. El sistema de *url routing* de la aplicación determina qué controlador y acción debe invocar y con qué parámetros.

3. El controlador realiza la mayor parte de la operación necesaria en el lado del servidor. Para ello puede utilizar uno o varios modelos para realizar operaciones sobre la base de datos y también una o varias clases

auxiliares en caso de que la operación lo requiera.

4. El controlador invoca la vista correspondiente, que es enviada al usuario y procesada por el navegador web. Esta vista contiene también código de front-end (Javascript), que añade funcionalidad en la interfaz de usuario.

El flujo alternativo de las peticiones asíncronas (Ajax) que se producen sin recargar la web es el siguiente:

1. El usuario utiliza un control de la interfaz de la aplicación que genera una petición HTTP asíncrona al servidor mediante el código Javascript.

2. Se realizan los pasos 2 y 3 del flujo normal.

3. El controlador puede devolver una vista, que será una porción de código HTML que contendrá una sección determinada de la página; o puede devolver simplemente datos en formato JSON.

4. El front-end de la aplicación recibe la respuesta del controlador y realiza la acción correspondiente:

   - Si la respuesta es código HTML, inserta dicho código en la sección correspondiente de la página.

   - Si la respuesta son datos, los utiliza como sea necesario, en la mayor parte de los casos, para poblar una estructura en el código HTML, o mostrar al usuario el resultado de una acción.

5.1.1. Detalles de la arquitectura

Url routing

El módulo de url routing es una de las utilidades básicas para la aplicación MVC, ya que permite definir un sistema en el que la url identifica el controlador, la acción a ejecutar y los parámetros de ésta.

Para realizar el url routing, son necesarios 2 recursos:

- Fichero de configuración de directorios del servidor (en este caso, el fichero .htaccess en el servidor Apache).
  Mediante este fichero se define el redireccionamiento de todas las peticiones

HTTP relativas a la ruta de la aplicación utilizando para ello reglas que se ejecutan por orden de aparición, hasta que una de ellas se cumpla:

- Si la url pretende acceder a un fichero existente y permitido, omitir el redireccionamiento y proporcionar dicho recurso (necesario para que desde la propia aplicación se puedan definir accesos a recursos necesarios como imágenes o scripts).
- En caso de no cumplir la condición anterior, redireccionar la petición al fichero PHP de la clase base *Application*.

- Fichero *index.php* y clase *Application*, realizan el mapeo de la url a la estructura de clases de la aplicación. Reciben la gran parte de las peticiones HTTP de la aplicación, que van destinadas a ejecutar una acción de un controlador específico.

  Inicialmente el script *index.php* realiza un procesamiento de la url, que debe cumplir la estructura específica http://<dominio>/ilmt/<controlador>/<acción>/<parámetros>y extrae los valores del controlador, acción y parámetros.

  A continuación ejecuta la función *loan_controller* de la clase *Application*, que realiza ciertas comprobaciones necesarias (comprueba el login del usuario, sus privilegios para ejecutar dicha acción, la existencia del controlador y la acción).

  Si las comprobaciones son satisfactorias, crea una instancia de la clase controlador y ejecuta la acción correspondiente. En caso de error, genera la respuesta correspondiente al usuario (redirección a la pantalla de login en caso de login inválido, redirección a la dirección base de la aplicación en caso de controlador no encontrado, etc).

  Ejemplo: *http://mipaginaweb.com/usuario/mostrar/id/1*

  Esta url resultaría en una llamada a la función *mostrar* del controlador *usuario* con el parámetro *id* igual a 1.

Estructura del proyecto

Los ficheros de la aplicación están organizados en una estructura de directorios estándar en una aplicación MVC.

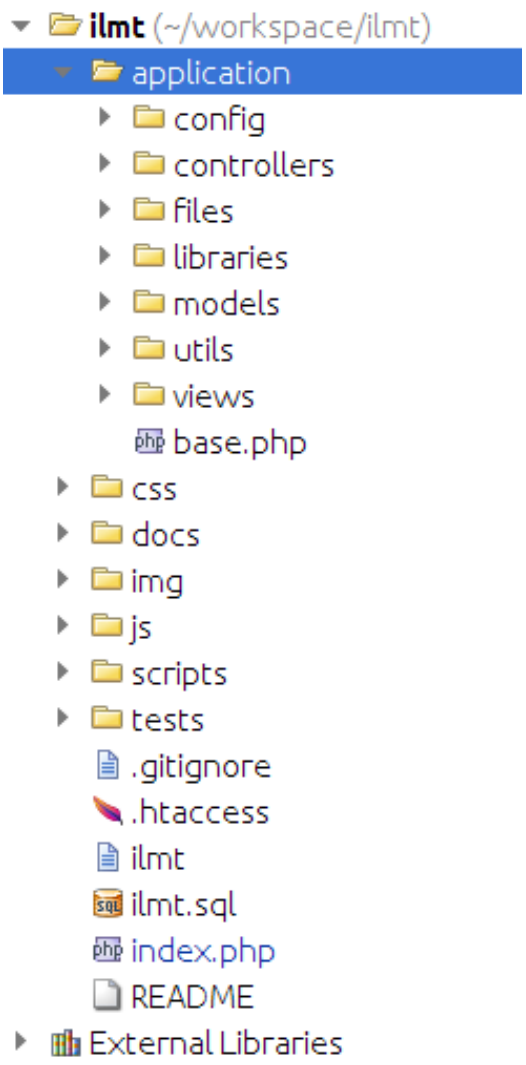

Figura 5.2: Estructura de directorios de la aplicación

El directorio raíz del proyecto contiene dos ficheros importantes: *index.php* y *.htaccess*. El fichero *index.php* contiene la entrada al sistema. Es un script PHP que se ejecuta el primero para cualquier petición al sistema. Realiza las siguientes tareas:

- Define ciertas variables globales necesarias, como el entorno de la aplicación (*APPLICATION_ENV*) y la ruta base de la aplicación (*BASE_PATH*).

- Importa los ficheros necesarios para el uso general de la aplicación, como *config.php* (contiene funciones que definen rutas necesarias para la aplicación y carga el Autoloader) y *Common.php* (define funciones de ayuda de uso general para la aplicación). Autoloader es un módulo que facilita la localización de los ficheros PHP donde se encuentran las clases referenciadas en el código.

- Realiza la descomposición en partes de la url y ejecuta el método *load_controller* de la clase *Application* para cargar el controlador y la acción correspondiente.

El fichero *.htaccess* especifica las reglas a seguir para el *Url mapping* explicado anteriormente en el apartado 5.1.1.

El directorio *application* contiene los ficheros principales de la aplicación y el grueso del proyecto.

- La carpeta *config* contiene ficheros necesarios para la configuración del sistema, como *application.ini* (parámetros de configuración), el fichero *autoloader.php* para la carga de las clases PHP, el fichero *Bootstrap.php* para la inicialización del sistema, y *config.php*, con funciones y variables globales de configuración.

- La carpeta *controllers* contiene todos los controladores de la aplicación.

- La carpeta *files* contiene los ficheros adicionales utilizados por la aplicación que no forman parte del código, principalmente imágenes.

- La carpeta *libraries* contiene las clases PHP que forman parte de las dependencias de la aplicación, pero que no definen en sí las funciones principales de la aplicación.

- La carpeta *models* contiene todos los modelos de la aplicación.

- La carpeta *utils* contiene aquellas clases que ofrecen pequeñas funciones de utilidad al resto de la aplicación y que pueden ser utilizadas en múltiples contextos.

- La carpeta *views* contiene todas las vistas de la aplicación, principalmente ficheros PHP que contienen todo el código HTML de la aplicación utilizando variables PHP definidas por los controladores que invocan cada vista. También incluyen fragmentos de código Javascript relativos a cada vista.

La carpeta *css* contiene todos los ficheros que definen el estilo CSS de la aplicación.

La carpeta *js* contiene todos los ficheros Javascript comunes a múltiples partes de la aplicación, de manera que su definición ha sido extraída a ficheros externos en lugar de encontrarse en las vistas correspondientes.

La carpeta *scripts* contiene múltiples scripts (ver definición en el glosario) PHP que cumplen diversos objetivos, como eliminar los datos de la base de datos para volver a un estado inicial, insertar datos iniciales necesarios para el funcionamiento de la aplicación, etc.

La carpeta *tests* contiene los tests de PHP definidos para algunos módulos de la aplicación.

Bootstrap

No confundir con el framework Twitter Bootstrap comentado anteriormente.

Esta clase tiene el cometido de inicializar el contexto en el que se debe ejecutar la aplicación. Cualquier operación que sea necesario realizar de manera previa a la ejecución normal de la aplicación, debe encontrarse en la clase Bootstrap.

Las dos funcionalidades principales que realiza el módulo Bootstrap son:

- Cargar en memoria el fichero de configuración *application.ini*
- Inicializar el adaptador de la base de datos (módulo PDO de PHP que provee la interfaz necesaria para tratar la base de datos, usada por los modelos de la aplicación).

Controladores

Las clases denominadas controladores son aquellas que realizan el grueso del procesamiento de las tareas en el sistema. La clase principal *Application* contiene funciones comunes utilizadas en múltiples controladores y especialmente una función clave para el funcionamiento del MVC, *load_controller*.

Esta función toma los parámetros que han sido extraídos previamente de la url y a partir de ellos interpreta el nombre del controlador, la acción a invocar y los argumentos que recibe la acción. Con esta información, instancia la

clase del controlador, e invoca la acción correspondiente con los argumentos recibidos.

El controlador en cuestión realizará el procesamiento correspondiente según la tarea que desempeñe, utilizando si es necesario otras clases del sistema. Por último invocará una vista, o devolverá una estructura de datos (este caso se aplica para algunas acciones que siempre serán invocadas mediante una llamada Ajax y cuya respuesta será procesada por el código Javascript en el front-end).

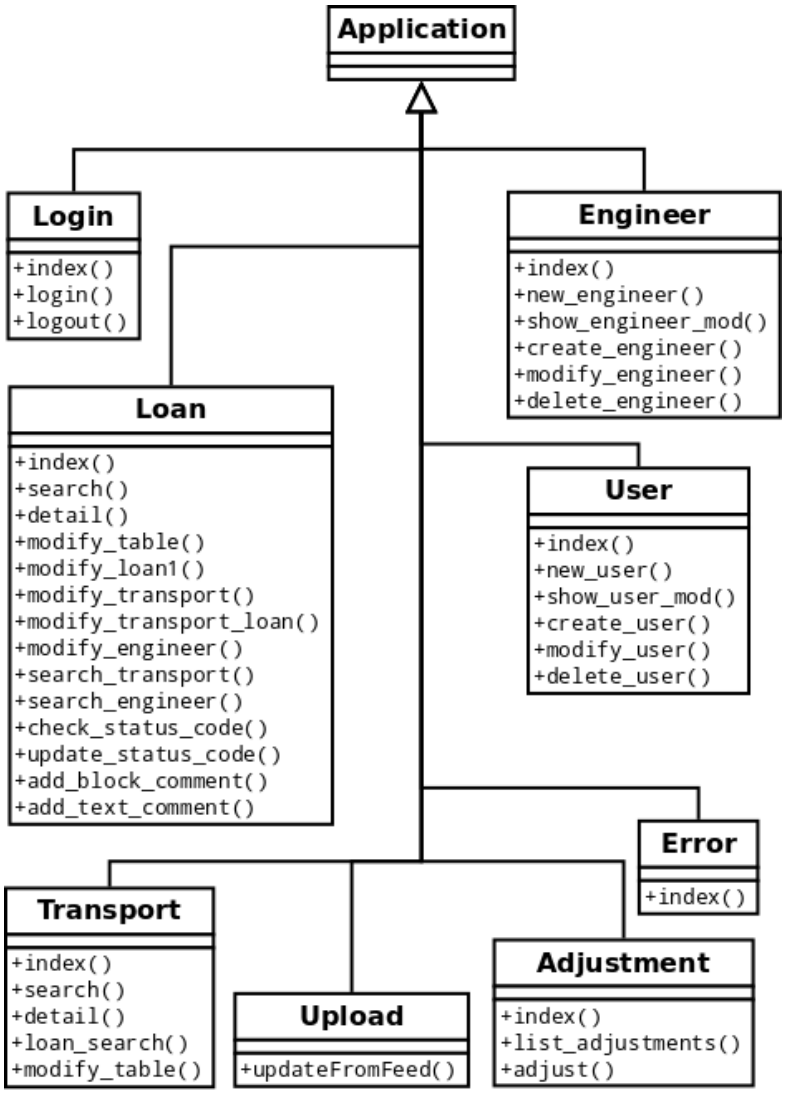

Figura 5.3: Diagrama de controladores

Modelos

Las clases denominadas modelos son aquellas que ofrecen una interfaz para realizar las operaciones necesarias sobre la base de datos de la aplicación. Todos los modelos de la aplicación se basan en la siguiente estructura:

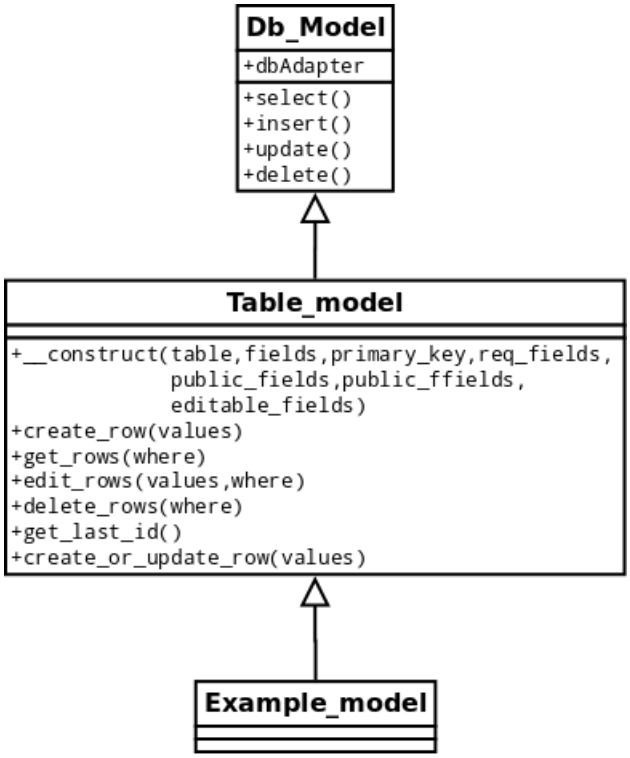

Figura 5.4: Diagrama de modelos de base de datos

Db_Model es la clase base de la capa de los modelos. Se construye sobre la interfaz PDO de PHP y define los aspectos básicos del acceso a base de datos de la aplicación.

Define las 4 funciones básicas CRUD (create, read, update, delete), ejecutando su contenido sobre el atributo *dbAdapter*, que es una instancia de la clase PDO de PHP.

Table_Model es la clase abstracta que representa los modelos de la aplicación para el acceso a tablas concretas de la base de datos. Se construye sobre la interfaz PDO de PHP y define los aspectos básicos del acceso a base de datos de la aplicación. PDO proporciona una capa de abstracción de acceso a datos de manera que, independientemente de la base de datos que se esté

utilizando, se pueden usar las mismas funciones para realizar las consultas y operaciones sobre los datos.

Al instanciarla es necesario especificar una serie de valores:

- Tabla a la que se refiere el modelo.
- Campos de la tabla.
- Campos requeridos.
- Campos públicos (que serán mostrados en algún momento en la interfaz de usuario).
- Nombre de los campos públicos en un formato amigable para el usuario (nombre de los campos que será mostrado en la interfaz).
- Campos editables por el usuario (para aquellas tablas cuya información es editada por el usuario, listado de los campos editables).

Además ofrece funciones más específicas para el tratamiento de tablas:

- createRow: encapsula la función *create*.
- getRows: encapsula la función *select*.
- editRows: encapsula la función *update*.
- deleteRows: encapsula la función *delete*.
- createOrUpdateRow: implementa una nueva funcionalidad que permite, suministrando a la función una clave identificativa y no primaria de la tabla, y los datos a insertar/actualizar, actualizar dicha fila en caso de que exista o crearla en caso de que no exista.

Estas funciones incluyen comprobaciones de las restricciones de cada tabla antes de realizar la operación correspondiente, basadas en los atributos antes definidos. (E.g. antes de realizar una inserción en la tabla, comprueba que los datos a insertar contienen todos los campos requeridos).

De forma general, por cada tabla de la base de datos se implementa una clase modelo, que hereda de *Table_model*, con la que se pueden realizar las operaciones básicas que define la clase padre y en caso de necesitar operaciones más complejas, se pueden definir como funciones adicionales en el modelo específico.

Los modelos de base de datos definidos en el sistema son los siguientes:

- Engineer_model
- User_model
- Adjustment_model
- Transport_model
- Loan1_model
- Loan2_model
- Comment_model
- Operation_model
- Rev_receipt_model
- Blind_receipt_model
- Handling_model

Se corresponden unívocamente con las tablas de la base de datos del mismo nombre (omitiendo el sufijo *_model*). Para mayor información consultar la descripción del modelo conceptual (apartado 4.1.1, página 24) y la especificación del modelo físico de datos (apartado 5.4.1).

Vistas

Las vistas son aquellos módulos del sistema que presentan la información del sistema al usuario, ofreciendo una interfaz de uso y definiendo la interacción que se puede realizar a través de la misma.

Las vistas son siempre generadas a través de una acción de un controlador, que realizará el procesamiento correspondiente y por último invocará una vista, transfiriendo a ésta los datos necesarios para la interfaz, siendo la vista quien define la manera de mostrar dichos datos.

Las vistas también contienen la mayor parte del código Javascript de front-end, ya que su principal función es ayudar a construir las funcionalidades de la interfaz de usuario.

Se componen principalmente de código HTML, incluyendo cláusulas de PHP para acceder a los datos necesarios definidos por el controlador y definen en muchos casos funciones Javascript.

5.2. Diseño de subsistemas

En esta sección se describen los subsistemas que componen la aplicación a nivel de detalles de las clases que forman el sistema.

En cada subsistema se especificarán los elementos de todas las capas que componen el subsistema (cada subsistema está definido de manera vertical a lo largo de las múltiples capas), incluyendo controlador, modelos, vistas y módulos adicionales en caso de haberlos.

Los subsistemas que componen el sistema son los siguientes:

- Login: Este subsistema sistema comprende la funcionalidad necesaria para realizar el login y logout del usuario en la aplicación.
- Componentes: Este subsistema comprende toda la funcionalidad relacionada con los componentes: búsqueda de componentes, visualización de detalles y edición de un componente.
- Transportes: Este subsistema comprende toda la funcionalidad relacionada con los transportes: búsqueda de transportes, visualización de detalles y edición de un transporte.
- Actualización de base de datos: Este subsistema comprende toda la funcionalidad relacionada con la actualización de la base de datos a partir de feeds.
- Ajustes: Este subsistema comprende toda la funcionalidad necesaria para la visualización de todos los componentes con peticiones de ajuste y realizar dichos ajustes.
- Ingenieros: Este subsistema comprende toda la funcionalidad necesaria para crear, editar y eliminar la información de los ingenieros en la aplicación.
- Usuarios: Este subsistema comprende toda la funcionalidad necesaria para crear, editar y eliminar la información de los usuarios en la aplicación.

5.2.1. Subsistema login

Controlador

El controlador utilizado es la clase *Login*. Sus funciones se describen a continuación:

- La función *index* únicamente carga la vista de la página principal de login.
- La función *login* es invocada cuando el usuario realiza *submit* del formulario de login, recibiendo como parámetros el nombre de usuario y contraseña.

  Realiza la validación del usuario y en caso de ser los datos válidos, redirige a éste a la página principal de la aplicación. En caso de ser inválidos, muestra un mensaje de error al usuario.

  Se utiliza la variable global de PHP $_SESSION para almacenar la información del usuario que hace login y posteriormente validar el login en las demás acciones.
- La función *logout* es invocada cuando el usuario presiona el botón de logout con el fin de salir de su sesión. En esta acción se destruye el contenido de la variable que almacena los datos de login del usuario y se redirige al usuario a la página inicial de login.

Modelos

El modelo utilizado en este subsistema es la clase *User_model*, de la que se utiliza la función *validate_user* para realizar la validación mediante nombre de usuario y contraseña.

Vistas

En este subsistema se utiliza una única vista, *login_view*, que ofrece la pantalla de login al usuario.

Utiliza el fichero Javascript *mod_create_user.js*, que contiene utilidades orientadas al login de usuario y creación y edición de usuarios, para realizar una validación previa del formulario de login (comprobar que contiene los datos necesarios: nombre de usuario y contraseña) antes de enviarlo al servidor para la validación

real.

5.2.2. Subsistema componentes

Este subsistema comprende toda la funcionalidad relacionada con los componentes: búsqueda de componentes, visualización de detalles y edición de un componente.

Controlador

El controlador utilizado es la clase *Loan*. Sus funciones se describen a continuación:

- Las funciones *index* y *search* contienen la funcionalidad de búsqueda de componentes. La primera función, *index*, se encarga de mostrar el formulario de búsqueda con todos los campos necesarios. La función *search* recibe la petición con los datos de la búsqueda y obtiene todos los componentes que cumplan el criterio utilizando el modelo correspondiente. Por último transfiere los resultados de la búsqueda a la vista correspondiente para que sean mostrados al usuario.
- La función *detail* muestra al usuario la página de detalles/edición de una componente. Para ello obtiene todos los datos de las tablas necesarias mediante los modelos correspondientes y carga la vista *loan_detail_view*.

El resto de las funciones se utilizan para llevar a cabo la edición y modificación de los detalles de un componente.

- Las funciones que comienzan con el prefijo *modify_* sirven para modificar filas específicas de las tablas de la base de datos que componen la información de un componente. Después de la modificación cargan la vista correspondiente que contendrá la información actualizada de la fila modificada.
- Las funciones *search_transport* y *search_engineer* realizan búsquedas en las tablas correspondientes de la base de datos. Estas funciones se utilizan al editar el ingeniero o uno de los transportes asociados a un componente, para asociar el resultado de dicha búsqueda al componente.
- Las funciones *check_status_code* y *update_status_code* llevan a cabo la actualización del código de estado de un componente. La función *check_status_code* define una lógica que, en base a la información

del componente, determina si debe cambiar el código. Esta función es invocada siempre que se realiza cualquier cambio en la información del componente.

- Las funciones *add_block_comment* y *add_text_comment* sirven para añadir comentarios a los detalles del componente. La primera es invocada automáticamente siempre que se realiza cualquier cambio en los detalles del componente y registra dicho cambio y el usuario que lo realiza. La segunda función permite al usuario añadir un comentario personal a los detalles del componente.

Modelos

Este subsistema utiliza múltiples modelos para manipular toda la información relacionada con los componentes. La información de un componente está distribuida en múltiples tablas de la base de datos y existe un modelo por cada tabla.

Los siguientes modelos permiten tratar la información de los componentes:

- Loan1_model
- Loan2_model
- Field_value_model
- Engineer_model
- Transport_model
- Handling_model
- Operation_model
- Rev_receipt_model
- Blind_receipt_model
- Adjustment_model
- Comment_model

Todos los modelos listados son necesarios para crear, modificar y eliminar la información de la que se compone un componente.

Vistas

Este subsistema utiliza múltiples vistas para mostrar la información relacionada con la búsqueda y edición de componentes.

En la página de búsqueda de componentes, se utilizan las siguientes vistas:

- loan_view
  Esta vista contiene la página con el formulario de búsqueda de componentes y el contenedor donde se insertará el resultado de las búsquedas, que inicialmente está vacío.

  Contiene además el código Javascript que realiza la llamada Ajax al pulsar el botón Buscar y carga el resultado de la búsqueda en el contenedor. Cuando el resultado de la búsqueda se carga en la página se muestra estructurado en una tabla y se activa la paginación para la tabla. Al seleccionar una página nueva de la tabla se carga nuevamente mediante Ajax el conjunto de filas que corresponden a la página seleccionada.

En la página de detalles y edición de un componente, se utilizan las siguientes vistas:

- loan_detail_view
  Esta vista contiene la página de detalles de un componente. Dicha página se divide en múltiples secciones, que son: Loan, Engineer, Transport, Handling, Operation, Receipt, Adjustment y Comment.

  Cada sección puede editarse por separado, haciendo click en el botón Edit o Add y salvando los cambios mediante el botón Save. Cada sección está contenida en una instancia de la vista block_view. Todo este proceso se realiza sin recargar la página, mediante llamadas Ajax. Para ello, se utiliza la vista table_detail_view, que se cargará cuando se salven los cambios con el contenido actualizado.

- block_view
  Esta vista contiene la estrutura de una sección que podrá ser editada de manera independiente.

- table_detail_view
  Esta vista se utiliza dentro de la vista block_view para definir la presentación de los datos dentro de cada sección, estructurados en una tabla.

- comment_detail_view
  Esta vista es un caso específico de block_view, para la sección comment (comentarios de un componente), cuya estructura difiere ligeramente del resto de las secciones.

5.2.3. Subsistema transportes

Este subsistema comprende toda la funcionalidad relacionada con los transportes, incluyendo su búsqueda, listado y edición.

Controlador

El controlador utilizado es la clase *Transport*. Sus funciones se describen a continuación:

- Las funciones *index* y *search* son análogas a las del controlador Loan. La primera función carga la vista inicial de la sección de transportes, mostrando el formulario de búsqueda. La segunda lleva a cabo la búsqueda con los datos enviados del formulario y devuelve una vista con el resultado.
- La función *detail* también es análoga a la del controlador Loan. Obtiene todos los detalles de un transporte utilizando los modelos correspondientes de la base de datos posteriormente carga la vista que muestra la página de detalles / edición del transporte específico.
- La función *modify_table* se utiliza para modificar los detalles del transporte. Después de actualizar la base de datos, carga la vista correspondiente con la información actualizada.

Modelos

Los modelos utilizados en este subsistema son los siguientes:

- Transport_model
  Este modelo se utiliza para la mayoría de las operaciones del controlador, que requieren mostrar y editar los datos de los transportes.
- Loan1_model

Este modelo es necesario para realizar la operación de búsqueda de todas los componentes vinculados a un transporte específico.

- Loan2_model
  Este modelo es necesario para realizar las actualizaciones necesarias en los componentes vinculados a un transporte, cuando éste es editado. (Ciertos cambios en los datos de un transporte pueden requerir actualizar la información de los componentes vinculados. Este vínculo se consulta a través de éste modelo).

Vistas

Este subsistema utiliza múltiples vistas para mostrar la información relacionada con la búsqueda y edición de transportes.

En la página de búsqueda de transportes, se utilizan las siguientes vistas:

- transport_view
  Esta vista contiene la página con el formulario de búsqueda de transportes, y el contenedor donde se insertará el resultado de las búsquedas, que inicialmente está vacío.

En la página de detalles y edición de un transporte, se utilizan las siguientes vistas:

- transport_detail_view
  Esta vista contiene la página de detalles de un transporte. Esta vista es análoga a *loan_detail_view* para los componentes. En el caso de los transportes, hay una única sección en la que se muestran los detalles de un transporte, y en la que que permite editarlos de manera idéntica a *loan_detail_view*.

  Esta vista incluye una funcionalidad adicional ofrecida en la interfaz a través del botón Mostrar componentes, que muestra una lista de todos los componentes vinculados al transporte al que refiere la página de detalles.

- block_view
  Esta vista contiene la estrutura de una sección que podrá ser editada de manera independiente.

- table_detail_view

Esta vista se utiliza dentro de la vista block_view para definir la presentación de los datos dentro de cada sección, estructurados en una tabla.

5.2.4. Subsistema actualización de base de datos

Este subsistema se encarga de toda la funcionalidad requerida para actualizar la base de datos de manera masiva utilizando feeds de datos, al margen de las actualizaciones específicas que puede realizar el usuario desde la interfaz.

Este subsistema proporciona la estructura para que se puedan implementar de manera sencilla módulos que procesen ficheros con un formato específico y con determinada información de componentes, transportes y/o ingenieros.

En este proyecto se ha implementado el módulo para la actualización de un fichero específico que contiene por cada entrada información básica del componente y del ingeniero relacionado con el componente.

Como futuro desarrollo quedaría implementar los módulos que permitan actualizar de nuevos ficheros que contengan información adicional de los componentes y de los transportes.

Controlador

El funcionamiento básico consiste en leer los datos de entrada de una serie de ficheros (feeds) que contienen los datos en un formato específico. A partir de estos datos se creará nueva información en la base de datos y se actualizará la información ya existente.

Además del controlador, se han utilizado una serie de clases auxiliares para facilitar esta funcionalidad, que se pueden observar en el siguiente diagrama (el diagrama recoge únicamente aquellos detalles de las clases considerados necesarios para la explicación):

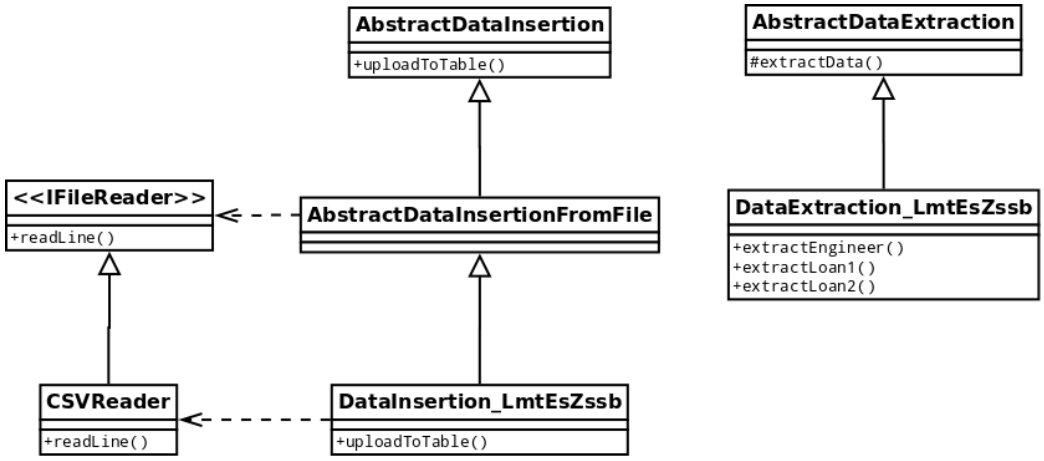

Figura 5.5: Diagrama de clases auxiliares del módulo Upload

- Se define la interfaz *IFileReader* para tratar los ficheros de entrada, con la función básica *readLine*, que obtendrá una línea nueva del fichero a leer. La clase *CSVReader* implementa dicha interfaz para ficheros con formato CSV.

- La clase *AbstractDataInsertion* define la interfaz para implementar la funcionalidad de insertar datos desde una fuente genérica de datos a una tabla de la base de datos. *AbstractDataInsertionFromFile* especifica más la clase anterior e implementa la funcionalidad genérica para insertar datos usando como origen un fichero de datos, por tanto, utiliza la interfaz *IFileReader*.

- La implementación para el fichero LmtEsZssb se encuentra en *DataInsertion_LmtEsZssb*. Esta clase define qué campos debe leer exactamente del fichero, y especifica que el fichero estará en formato CSV, por tanto el *IFileReader* utilizado será una instancia de *CSVReader*

- Por último, una vez los datos del fichero están almacenados en la tabla correspondiente siguiendo un formato similar al del fichero (nombres de los campos, sin normalizar, etc.), el último módulo permite extraer de cada fila de dicha tabla la información requerida en el formato necesario con el que se insertará en las tablas que utiliza realmente el resto de la aplicación.
  AbstractDataExtraction proporciona la funcionalidad común para extraer datos de cada fila, y la especificación *DataExtraction_LmtEsZssb* implementa la funcionalidad específica para el fichero LmtEsZssb, del que se extraerá la información que se insertará en las tablas Loan1, Loan2 y Engineer.

El controlador Upload pone en común todas estas clases auxiliares y define una función por cada fichero del que se permite actualizar.

Cada una de estas funciones sigue un algoritmo similar:

1. Crear el módulo DataInsertion correspondiente al fichero que se va a leer, y mediante éste, leer toda la información del fichero e insertarla en la tabla correspondiente.
   E.g. Cargar la información del fichero LmtEsZssb en su tabla, manteniendo el mismo formato del fichero, una fila por cada entrada del fichero.

2. Leer de la tabla, y por cada fila, usar el módulo específico de DataExtraction para obtener todos los bloques de datos específicos a las tablas de la aplicación.
   E.g. Usar DataExtraction_LmtEsZssb para extraer de la información de la fila los datos de Loan1, Loan2 y Engineer.

3. Tratar cada bloque de manera individual, insertando o actualizando la tabla según corresponda.
   E.g. Actualizar o crear, según corresponda, las filas en las tablas Loan1, Loan2 y Engineer.

La decisión de diseño de utilizar tablas intermedias para volcar el contenido de los feeds y posteriormente leer de ellas para actualizar la base de datos está motivada principalmente por aumentar la flexibilidad y extensibilidad del sistema.

Otra alternativa habría sido leer directamente los datos de los feeds y actualizar directamente las tablas de la base de datos a medida que se van consumiendo los ficheros.

Sin embargo, la alternativa implementada proporciona las siguientes ventajas de cara a futuras modificaciones y/o mejoras:

- Posibilidad de definir un campo de log en el que escribir un mensaje de error en caso de fallo al leer una entrada del feed.

- Posibilidad de definir un campo de hash (ver definición en el glosario) para optimizar el proceso, de manera que solo se copian del feed a la tabla aquellas entradas no existentes previamente o que presenten cambios en los datos.

- Mayor aislamiento de los diferentes procesos en caso de error.

La función que define el controlador Upload para actualizar la base de datos por cada feed utiliza dos funciones definidas en la clase *Application*: *getLock* y *releaseLock*. Estas funciones permiten obtener un *lock*, que es un mecanismo de sincronización para limitar el acceso concurrente a un mismo recurso.

En este caso se utiliza para evitar que múltiples usuarios puedan iniciar de manera paralela el proceso de actualización desde un mismo feed, ya que sólo debe haber un proceso realizando la actualización en un momento determinado. Cuando un segundo usuario intenta iniciar la actualización que ya está en marcha, la función del controlador falla al obtener el *lock*, y devuelve al usuario un mensaje de error, indicando que la actualización ya está en curso.

Modelos

Los modelos utilizados en este subsistema son todos los necesarios para realizar las actualizaciones en la base de datos:

- Loan1_model
- Loan2_model
- Engineer_model
- Transport_model
- Handling_model
- Operation_model
- Rev_receipt_model
- Blind_receipt_model

Vistas

Este subsistema únicamente necesita una única vista, ya que su funcionalidad se concentra principalmente en el lado del servidor:

- Upload_view
  La vista ofrece al usuario una lista de los feeds desde los que puede

actualizar, controles para iniciar dichas actualizaciones, y contiene una sección destinada a los resultados.

Al iniciar una actualización, se realiza la petición correspondiente al servidor de manera asíncrona, y cuando la actualización termina, el resultado de la operación se inserta en la página en el contenedor destinado a ello.

5.2.5. Subsistema ajustes

Este controlador comprende la funcionalidad necesaria para realizar los ajustes de componentes, proceso que consiste en marcarlos con el fin de que pasen a un estado en el que ya no se intentan recuperar.

Controlador

El controlador utilizado es la clase *Adjustment*. Sus funciones se describen a continuación:

- La función *index* muestra la vista principal del controlador, que ofrecerá la lista de componentes pendientes de ajuste (obtenida a su vez como resultado de la llamada a la función *list_adjustments*) y los controles necesarios para seleccionar elementos de la lista y posteriormente aprobar o rechazar dichos ajustes.
- La función *adjust* es invocada al enviar desde el frontend la petición de aprobar o rechazar los componentes a ajustar seleccionados de la lista. Utiliza el modelo correspondiente de base de datos para modificar todos los componentes seleccionados, insertando el valor del estado del ajuste (aprobado o rechazado), y por último recarga la vista que contiene la lista actualizada de componentes pendientes de ajuste.

Modelos

Los modelos utilizados en este subsistema son los siguientes:

- Adjustment_model
  Este modelo se utiliza para obtener el listado de todos los peticiones

de ajuste pendientes.

- Loan1_model
  Este modelo es necesario para obtener la información de los componentes vinculados con las peticiones de ajuste.

Vistas

Este subsistema utiliza una única vista:

- Adjustment_view
  La vista ofrece al usuario una lista de las peticiones de ajuste pendientes. Esta lista es básicamente una lista de componentes similar a la que se muestra como resultado de la búsqueda de componentes.

  Los controles incluyen un checkbox por cada componente, y un checkbox general (marcar / desmarcar todos) mediante el que seleccionar los componentes sobre los que se quiere realizar la acción.

  Además ofrece los botones Aprobar y Rechazar, acciones que se aplican a los componentes seleccionados en el momento de pulsar el botón, incluyendo un diálogo de confirmación de la acción.

  Tras realizar la acción, la lista de peticiones pendientes se recarga de manera asíncrona mediante una petición al servidor.

5.2.6. Subsistema ingenieros/usuarios

Los subsistemas ingenieros/usuarios son muy similares, ya que se basan en la misma estructura aunque con ciertas variaciones en los datos a tratar.

Proporcionan la funcionalidad necesaria para realizar la gestión de los ingenieros y usuarios. Permiten crear, editar y eliminar ingenieros/usuarios de la base de datos.

Por ello, se hará la especificación de los dos subsistemas dentro de la misma sección, con el fin de evitar redundancias.

Controlador

Los controladores utilizados para los subsistemas son *Engineer* y *User*. Definen las siguientes funciones:

- La función *index* muestra la lista de todos los ingenieros/usuarios existentes en la base de datos. Proporciona los controles necesarios para editar o borrar cada uno de las entradas que aparecen en la lista. Esta función carga la vista principal que se mostrará para todas las acciones que se ejecuten del controlador. Las demás acciones serán invocadas mediante peticiones *Ajax* y las vistas que generen como resultado se insertarán en la vista principal dentro de los contenedores html correspondientes.

- La función *new_engineer*/*new_user* muestra el formulario donde introducir los datos necesarios para crear un nuevo ingeniero/usuario.

- La función *show_engineer_mod*/*show_user_mod* es análoga a la anterior. Es invocada al hacer click en el botón para editar un ingeniero/usuario. Carga el mismo formulario que la función anterior, salvo por el añadido de que inserta los datos del ingeniero/usuario a editar en los campos del formulario.

- La función *create_engineer*/*create_user* es invocada cuando se realiza el submit en el formulario de creación de ingeniero/usuario. Dicho formulario se valida en el frontend mediante código Javascript antes de ser enviado al servidor, y posteriormente se verifica nuevamente en el controlador, comprobando que se han recibido todos los valores necesarios. A continuación inserta la información del nuevo ingeniero/usuario en la base de datos, utilizando el modelo correspondiente, y por último, en función del resultado, muestra un mensaje de éxito o de error.

- La función *modify_engineer*/*modify_user* es análoga a la función anterior. De una manera similar, obtiene el resultado del submit del formulario de edición de un ingeniero/usuario, y realiza la llamada al modelo para editar la fila correspondiente en la base de datos. Por último muestra el resultado de la modificación.

- La función *delete_engineer*/*delete_user* es invocada al hacer click en el botón para borrar un ingeniero/usuario. Realiza la llamada al modelo correspondiente para eliminar la fila de la base de datos, y devuelve la vista que contiene la lista actualizada de ingenieros/usuarios.

Modelos

Los modelos utilizados son *Engineer_model* y *User_model* respectivamente, para mostrar, editar y eliminar la información correspondiente.

Vistas

Las vistas utilizadas son:

- engineer_view / user_view
  Esta es la vista principal de esta sección y funciona como contenedor de las demás vistas. Básicamente contiene 2 secciones donde se cargarán las otras vistas: la sección de creación o edición de un ingeniero/usuario, y la sección que contendrá la lista de ingenieros/usuarios. Muestra la lista de ingenieros/usuarios con los controles para crear, editar o eliminar un ingeniero/usuario.

- engineer_list_view / user_list_view
  Esta vista contiene una tabla que almacenará la lista de los ingenieros/usuarios del sistema, mostrando sus datos por filas, además de ofrecer los controles para editar o eliminar cada una de las entradas de la tabla.

- engineer_mod_view / user_mod_view
  Esta vista contiene el formulario de creación o edición de un ingeniero/usuario, se cargará en la vista principal en caso de pulsar los botones correspondientes de creación o edición.

5.3. Diseño de la interfaz de usuario

En este apartado se muestran las distintas pantallas que componen la interfaz de usuario del sistema.

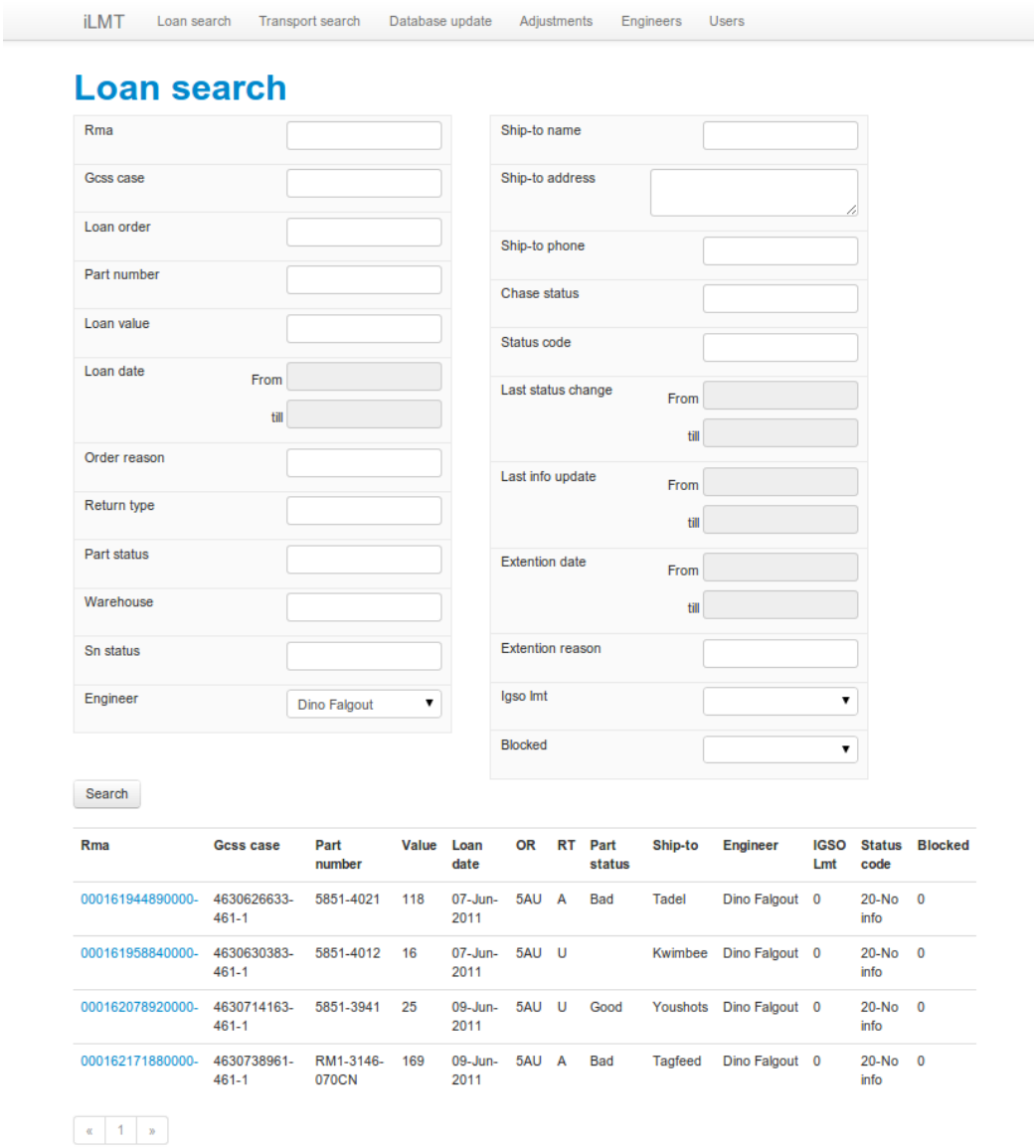

Figura 5.6: Interfaz: búsqueda de componente

En la figura 5.6 se muestra la interfaz de búsqueda de componente. La interfaz contiene el formulario de búsqueda en que el usuario debe insertar sus criterios de búsqueda mediante los que quiere filtrar los componentes existentes en la base de datos.

Al realizar la petición de búsqueda mediante el botón de submit, el resultado se carga en la parte inferior de la interfaz, insertando los datos en una tabla con paginación.

El valor *RMA* de cada componente es un enlace a la página de detalles de ese componente específico.

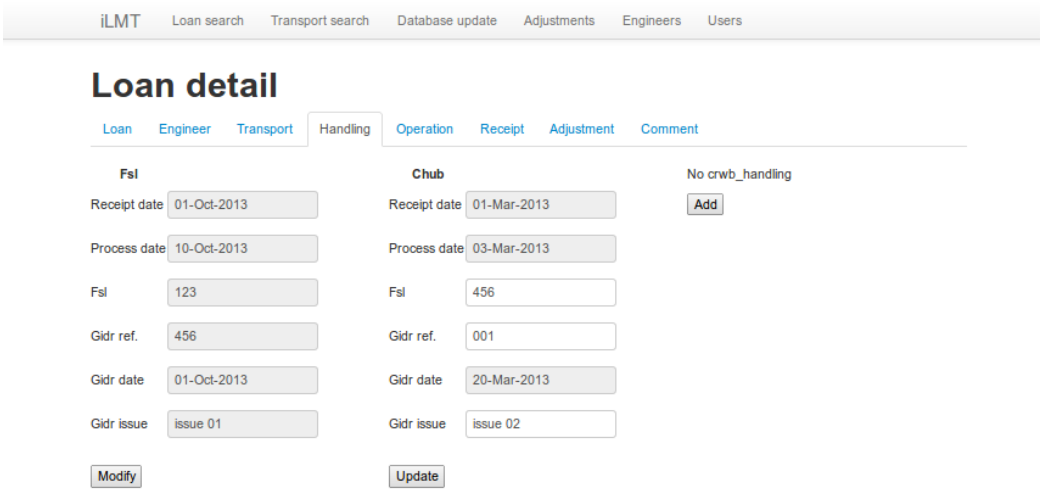

Figura 5.7: Interfaz: detalles de un componente

En la figura 5.7 se muestra una pantalla de la interfaz de edición de detalles de un componente. En este caso está seleccionada la sección *Handling*.

Muestra los formularios correspondientes a los datos de la sección para el componente al que refieren los detalles. Los botones permiten modificar y actualizar cada módulo de información por separado sin recargar la página.

Cada vez que el usuario pulsa el botón *Update*, se realiza una petición asíncrona al servidor, que actualiza los datos en la base de datos, y recarga la sección de la página correspondiente mostrando los datos actualizados.

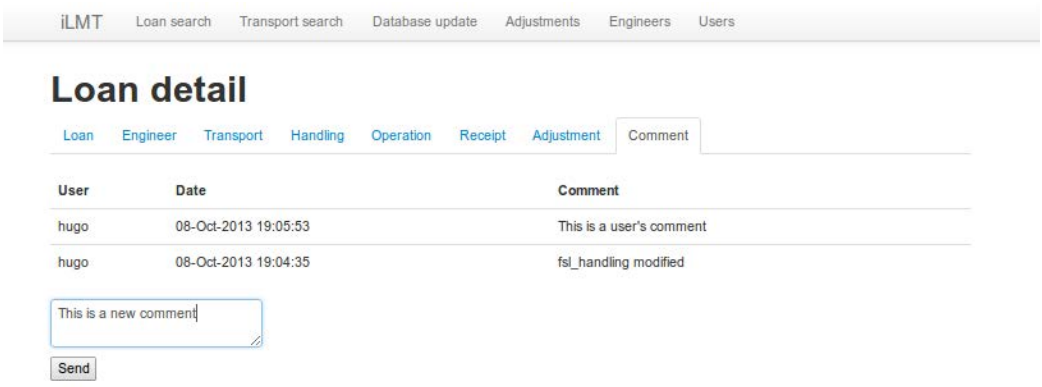

Figura 5.8: Interfaz: detalles de un componente, edición de comentarios

En la figura 5.8 se muestra la pantalla de la interfaz de edición de comentarios de un componente. Los comentarios muestran el registro automático de modificaciones realizadas por los usuarios en el componente, y los comentarios introducidos por los usuarios, ordenados por fecha y hora.

Debajo de la lista de comentarios se encuentra el formulario para introducir un comentario nuevo. Al guardar el comentario, se actualiza automáticamente la lista de comentarios con los cambios.

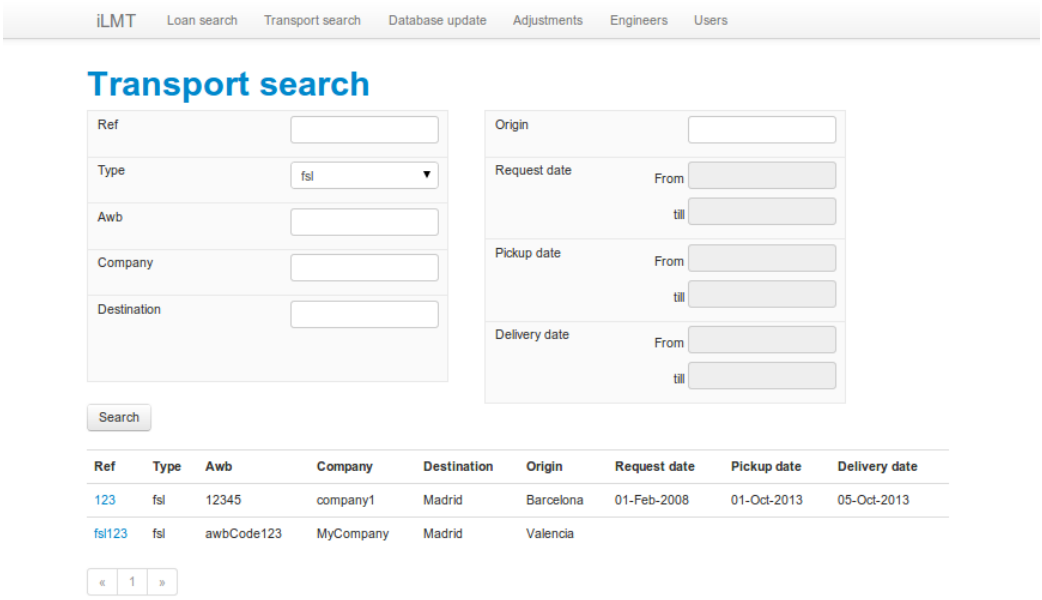

Figura 5.9: Interfaz: búsqueda de transporte

En la figura 5.9 se muestra la interfaz de búsqueda de transporte. Contiene el formulario con los campos a rellenar para realizar la búsqueda.

La parte inferior de la interfaz muestra el resultado de la búsqueda, en forma de tabla paginada que contiene los datos de los transportes.

El valor *Ref* de cada transporte es un enlace a la página de detalles de ese transporte específico.

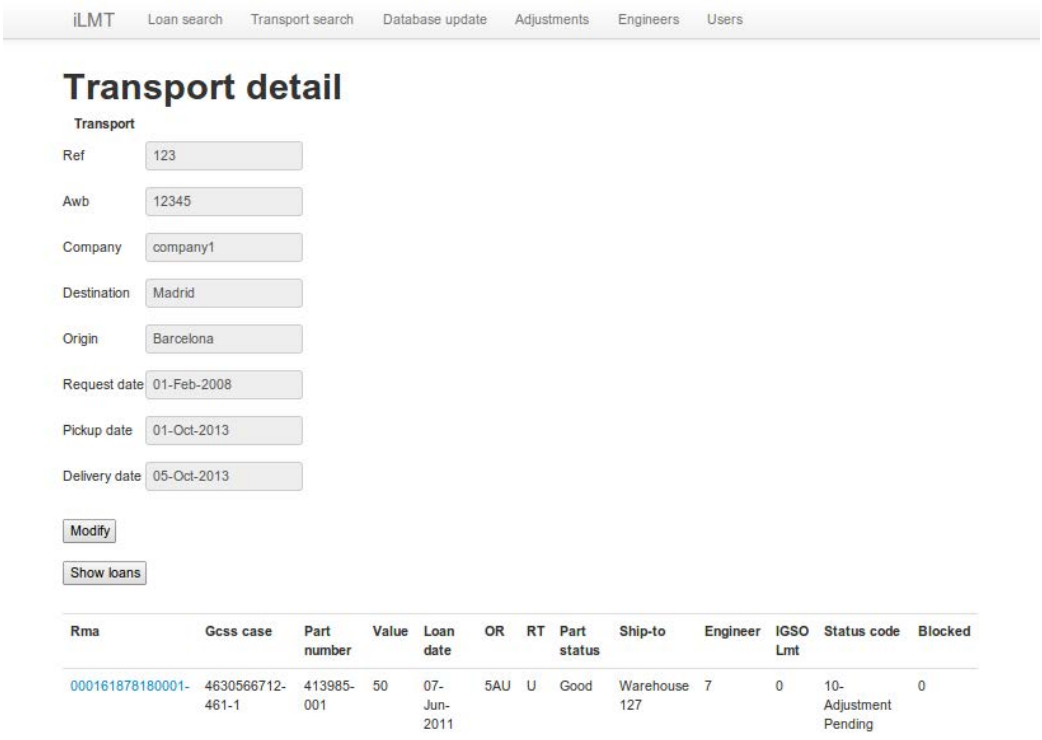

Figura 5.10: Interfaz: detalles de un transporte

En la figura 5.10 se muestra una pantalla de la interfaz de edición de detalles de un transporte.

Muestra el formulario con los datos del transporte al que refieren los detalles. El botón permite modificar y actualizar los datos del transporte sin actualizar la página.

Cada vez que el usuario pulsa el botón *Update*, se realiza una petición asíncrona al servidor, que actualiza los datos en la base de datos, y recarga la sección que contiene los datos del transporte mostrando los datos actualizados.

El botón inferior *Show loans* muestra una tabla con los componentes vinculados al transporte en cuestión, de manera que se puede acceder fácilmente a los detalles de los mismos.

Las capturas a continuación muestran la interfaz de actualización de la base de datos a partir de un feed:

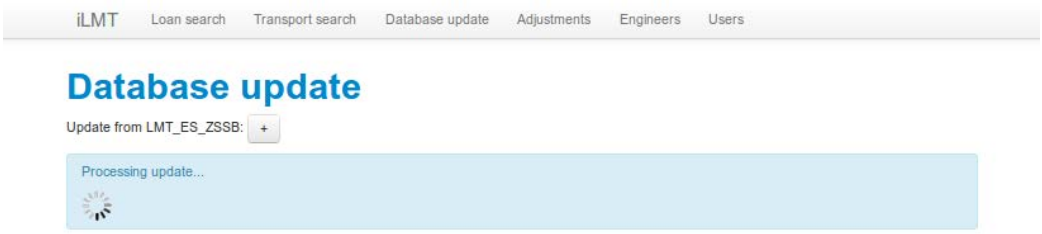

Figura 5.11: Interfaz: actualización de base de datos

En la figura 5.11 se muestra la pantalla de carga después de pulsar el botón correspondiente para actualizar de un feed específico, hasta que la actualización finaliza.

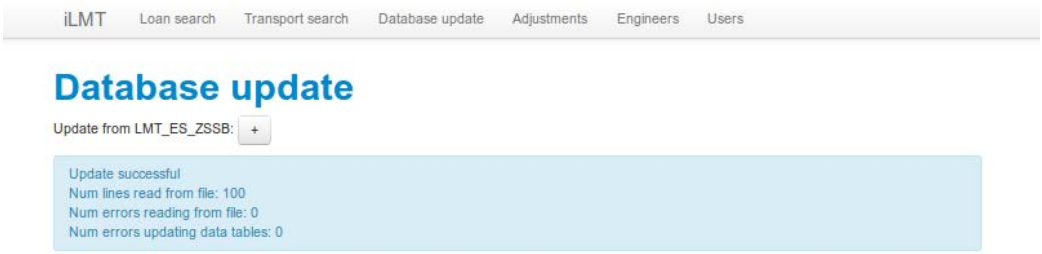

Figura 5.12: Interfaz: actualización de base de datos con éxito

En la figura 5.12 se muestra el resultado de la actualización, en caso de finalizar con éxito. Indica el número de líneas leídas del feed, número de errores leyendo del feed, y número de errores al actualizar la base de datos.

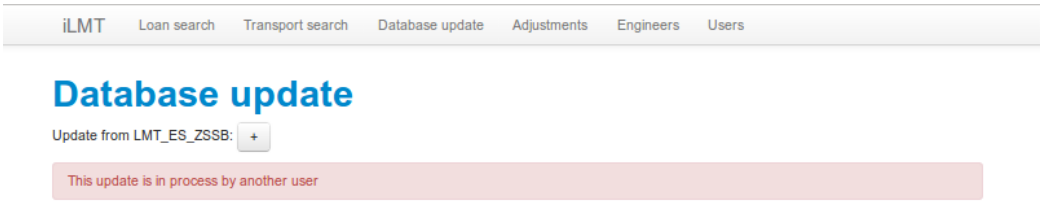

Figura 5.13: Interfaz: error en actualización de base de datos

En la figura 5.13 se muestra el resultado de ejecutar una actualización que ya ha sido iniciada por otro usuario. Gracias al sistema de lock, el usuario que ejecuta la actualización ya iniciada recibe un mensaje de error indicando que el proceso ya está siendo ejecutado por otro usuario.

Las capturas a continuación muestran la interfaz de edición de ingeniero y listado de los mismos:

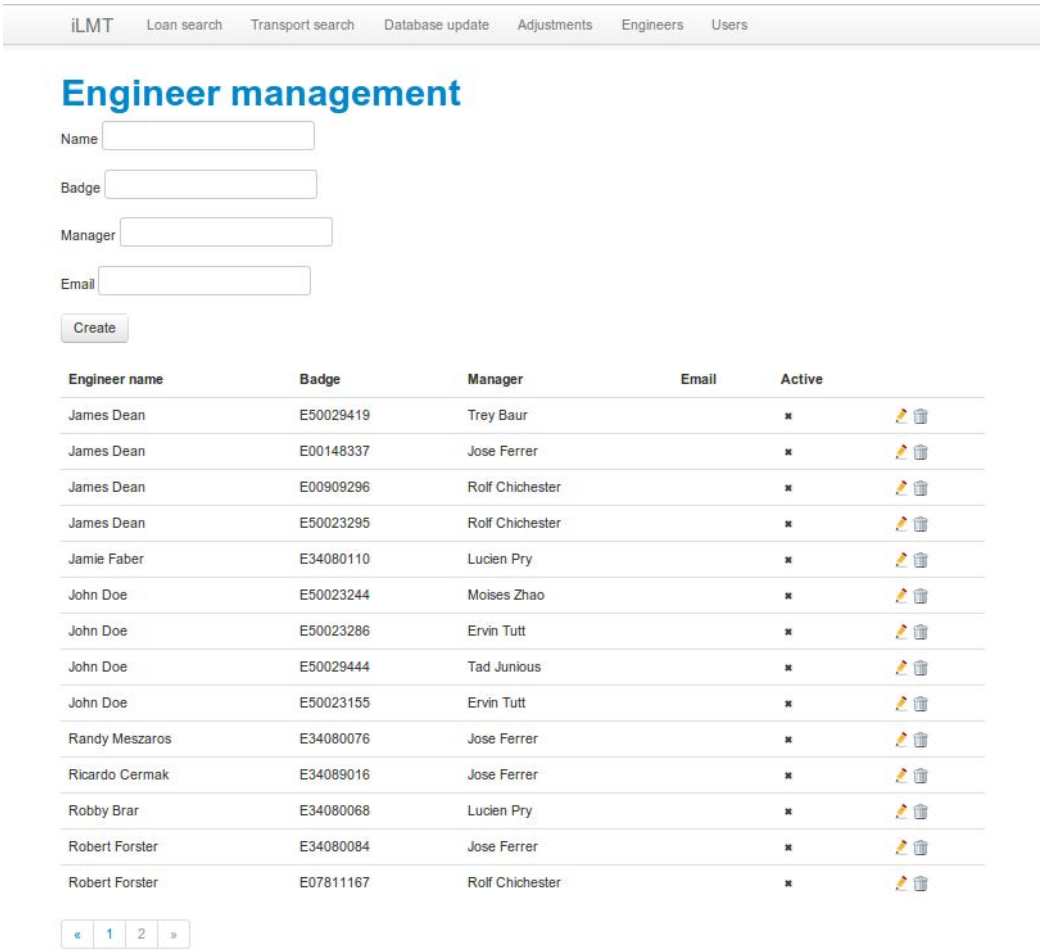

Figura 5.14: Interfaz: creación de ingeniero

Como se puede observar en la figura 5.14, la sección inferior contiene la lista de los ingenieros de forma permanente. La sección superior contiene el formulario de creación / edición de un ingeniero, que ha sido cargado en la página de forma asíncrona a continuación de haber pulsado el botón correspondiente para crear un nuevo ingeniero, o haber pulsado el botón de edición de un ingeniero.

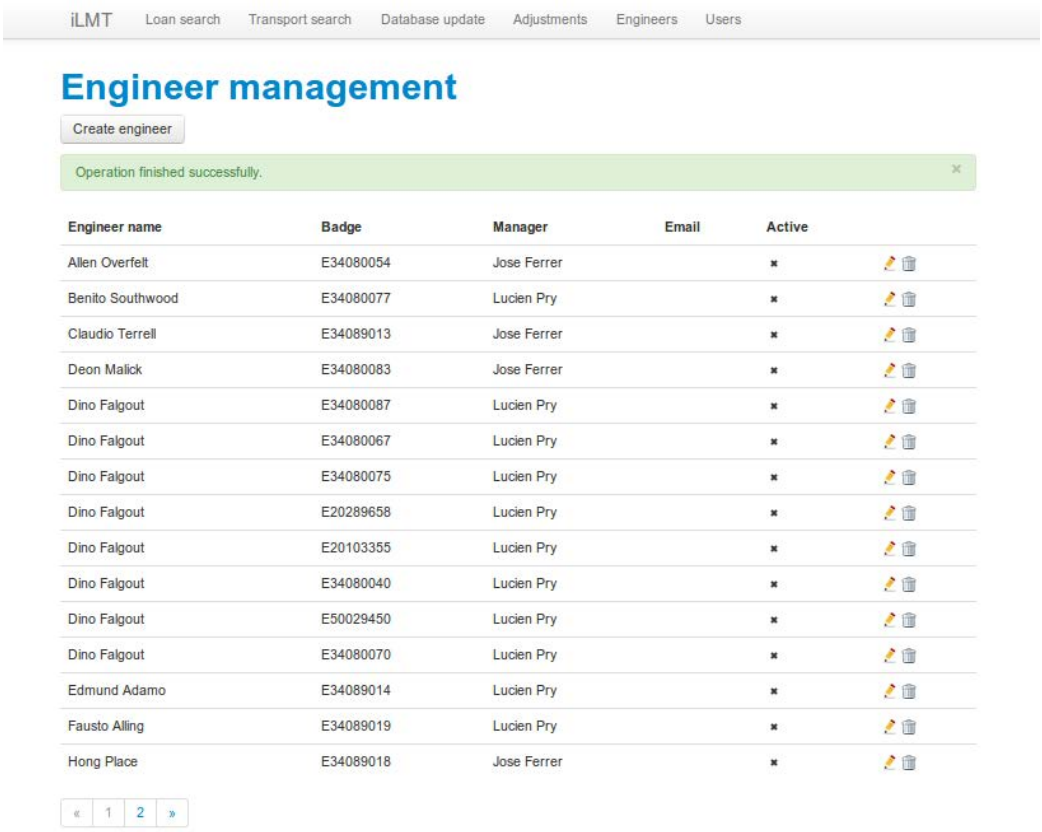

Figura 5.15: Interfaz: creación de ingeniero con éxito

Como se muestra en la figura 5.15, al pulsar el botón *Create* para crear un ingeniero, si la operación concluye con éxito, se muestra al usuario un mensaje de resultado satisfactorio y se recarga la lista de ingenieros de forma asíncrona con la información actualizada.

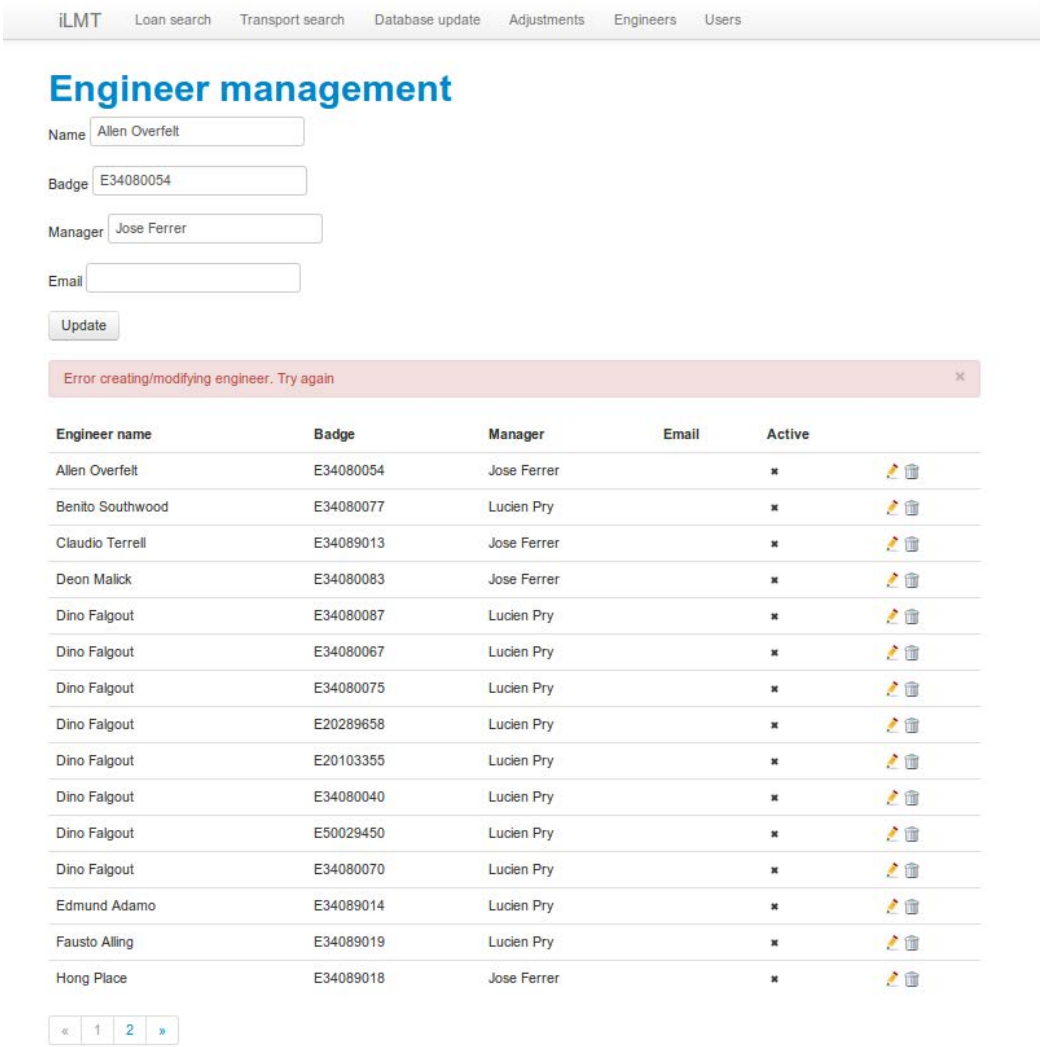

Figura 5.16: Interfaz: error en la creación de ingeniero

Como se muestra en la figura 5.16, si el resultado de la operación es erróneo, se muestra un mensaje de error al usuario y se mantiene el formulario con los datos para que el usuario pueda corregir los datos.

Las capturas a continuación muestran la interfaz de edición de usuario y listado de los mismos:

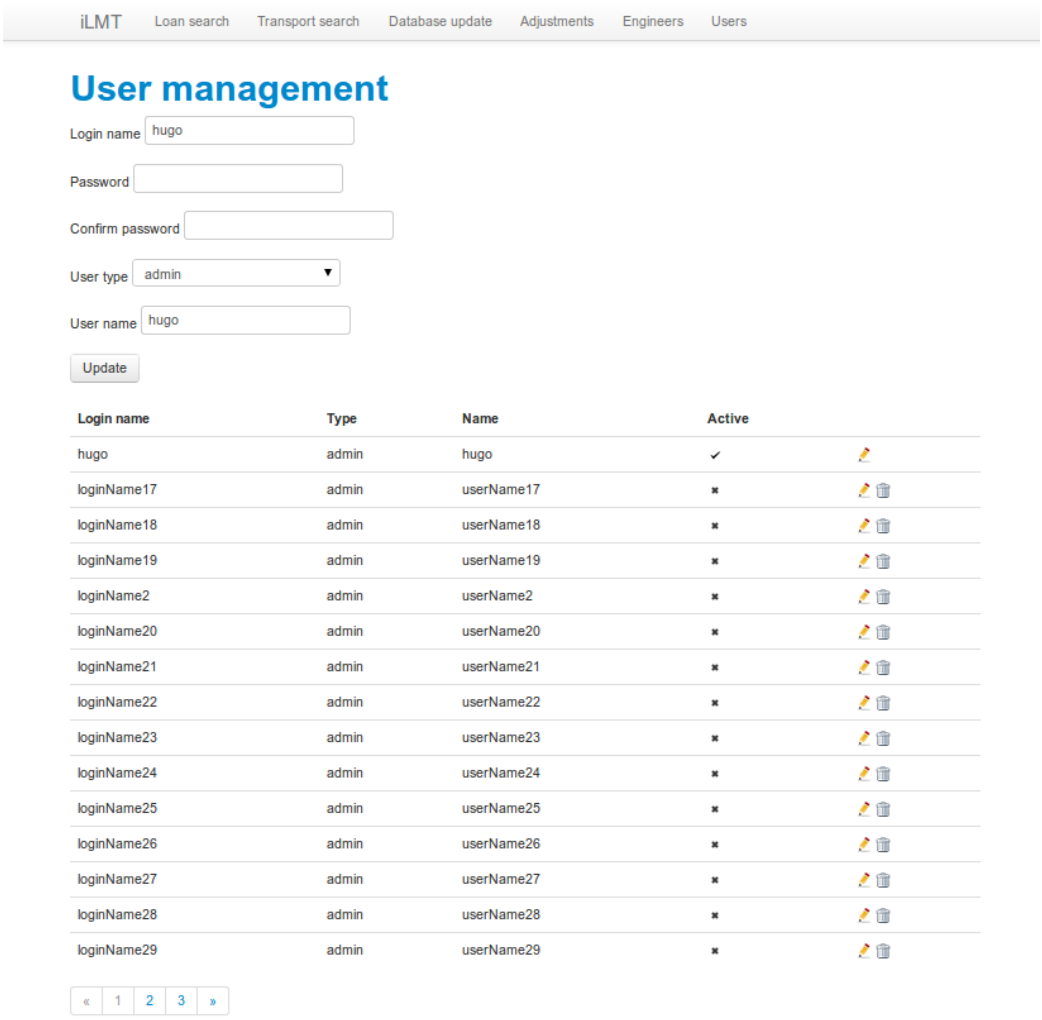

Figura 5.17: Interfaz: modificación de usuario

Como se puede observar en la figura 5.17, la sección inferior contiene la lista de los usuarios de forma permanente. La sección superior contiene el formulario de creación de un usuario, que ha sido cargado en la página de forma asíncrona a continuación de haber pulsado el botón correspondiente para crear un nuevo usuario, o haber pulsado el botón de creación de un nuevo usuario.

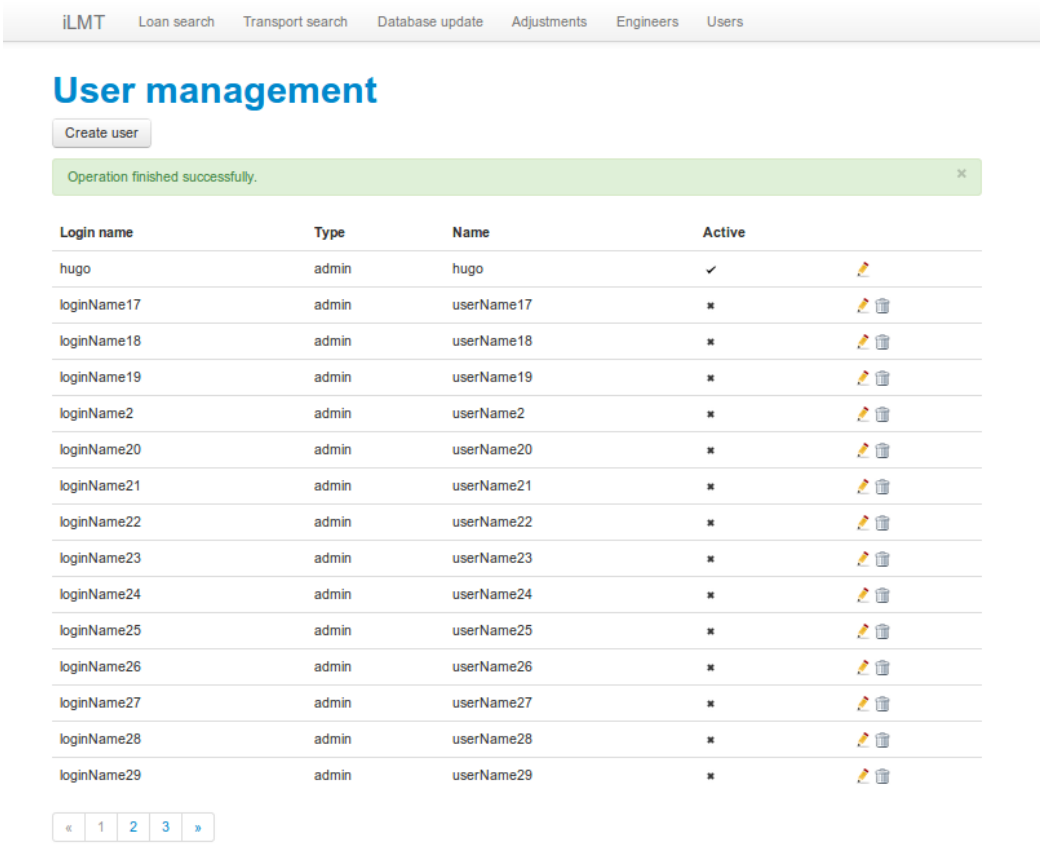

Figura 5.18: Interfaz: modificación de usuario con éxito

Como se muestra en la figura 5.18, al pulsar el botón *Create* para crear un usuario, si la operación concluye con éxito, se muestra al usuario un mensaje de resultado satisfactorio y se recarga la lista de usuarios de forma asíncrona con la información actualizada.

5.4. Modelo físico de datos

En esta sección se define la estructura física de datos que utilizará el sistema. También se analizan los caminos de acceso a los datos utilizados por cada clase del sistema.

5.4.1. Modelo físico de datos: tablas

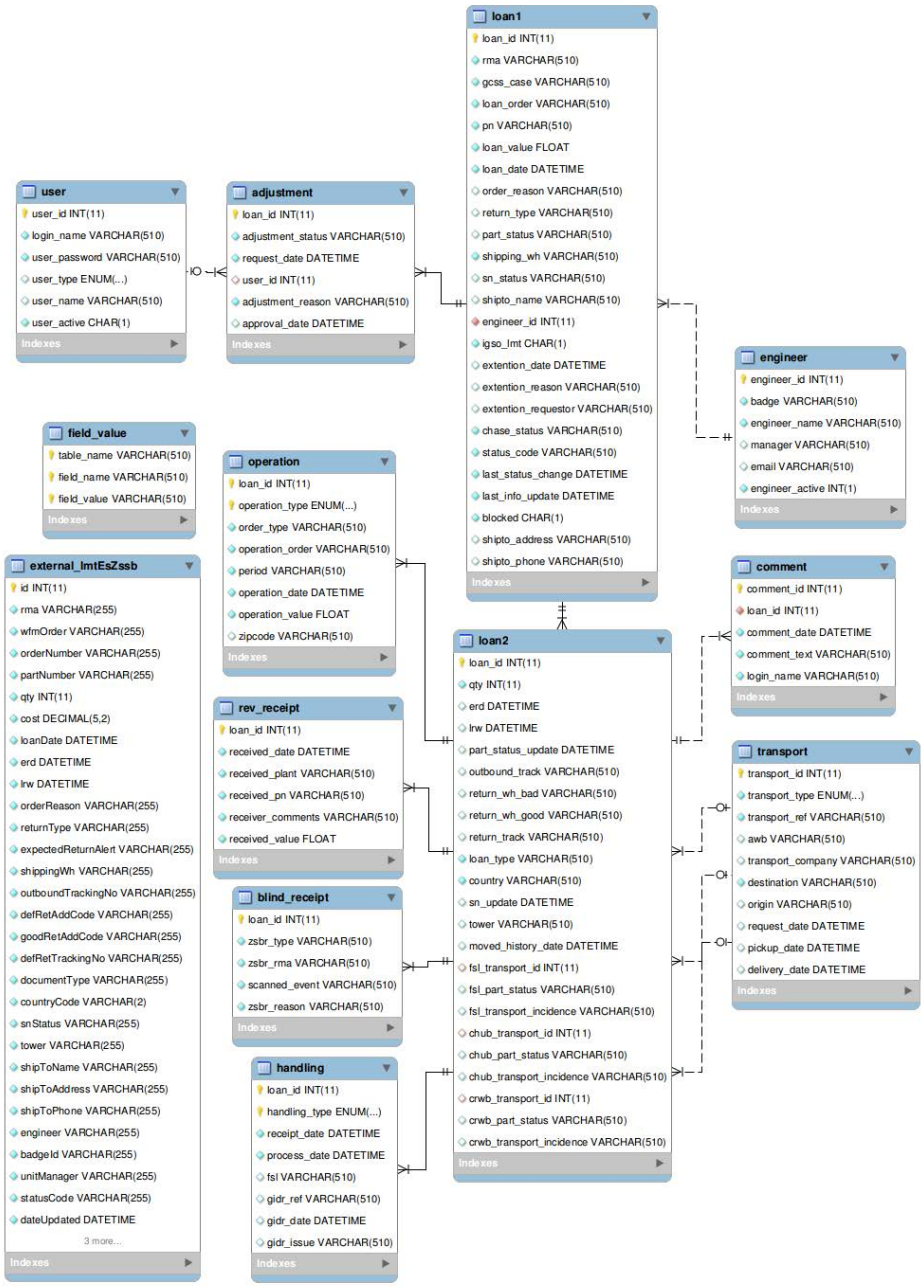

Figura 5.19: Diagrama de la base de datos

La estructura de la base de datos tiene el cometido de almacenar de manera estructurada la información necesaria de la aplicación, que se puede englobar

bajo los siguientes conceptos:

Componentes

La información se almacena en las tablas *loan1*, *loan2*, *operation*, *rev_receipt*, *blind_receipt*, *handling*, *adjustment*, *comment*.

Las tablas *loan1* y *loan2* almacenan la información principal de un componente. La razón de la división de esta lista de campos en dos tablas se basa en la eficiencia del acceso a dichas tablas. La tabla *loan2* está vinculada a *loan1* a través de la clave primaria y ajena *loan_id*.

La tabla *loan1* contiene todos los campos que pueden ser utilizados en la búsqueda de componentes. El hecho de separar esta información de la información no utilizada en la búsqueda y enlazarla mediante la misma clave primaria (*loan_id*), hace que la tabla en la que se realiza la búsqueda sea de menor tamaño, aligerando las operaciones sobre ella. Esta tabla contiene además un enlace como clave ajena a la tabla *engineer* a través del campo *engineer_id*.

La tabla *loan2* contiene además 3 vínculos como clave ajena a la tabla *transport*, que representan los 3 posibles enlaces de un componente a los 3 tipos correspondientes de transporte (*fsl*, *chub*, *crwb*), cuya información está almacenada en la misma tabla.

Las tablas *operation*, *rev_receipt*, *blind_receipt*, *handling* almacenan información adicional del componente, que puede existir o no, pero en caso de existir, dicha información se compone de básicamente todos los campos de la tabla correspondiente. Es por esta razón por lo que se definen como entidades independientes a la información principal del componente. Están vinculados al componente utilizando *loan_id* como clave ajena y primaria a la vez.

Para una descripción de los elementos listados previamente que componen la información de un componente, consultar la descripción de las entidades que forman un componente. (Apartado 4.1.1, página 24).

La tabla *adjustment* almacena información acerca de los ajustes realizados a los componentes. Es información opcional, ya que sólo un porcentaje de los componentes son ajustados.

Las tabla *comment* almacena dos tipos de información acerca de los componentes, bajo la misma estructura:

- Comentarios que los usuarios realizan en los componentes.
- Registro de los cambios que realizan los usuarios en los componentes, almacenados por la propia aplicación.

Transportes

La información se almacena en la tabla *transport*. Se compone de todos los datos relativos a un transporte, de forma independiente a los componentes con los que esté vinculado. Puede ser de 3 tipos (*fsl*, *chub*, *crwb*), almacenando la misma información en los 3 casos.

Ingenieros

La información se almacena en la tabla *engineer*. Contiene los datos identificativos y de contacto del ingeniero.

Usuarios

La información se almacena en la tabla *user*. Contiene los datos identificativos del usuario, necesarios para realizar el login, y además el grado de privilegios de éste.

Datos adicionales

La tabla *field_value* contiene la especificación de los valores posibles para ciertos campos de la base de datos. La decisión de crear una tabla aparte con esta finalidad tiene el objetivo de facilitar la modificación y creación de valores para cualquier campo en cualquier tabla de la base de datos.

5.4.2. Listado de campos

A continuación se definen dos listas de campos referenciadas en los requisitos del problema.

La lista de campos de búsqueda determina aquellos campos de la base de datos que deben ser incluidos en los campos de búsqueda del sistema, en la sección de búsqueda correspondiente (componentes y/o transportes). A continuación se listan los campos en la tabla 5.1, agrupados por las tablas de la base de datos a las que pertenecen.

loan1	**engineer**	**transport**
rma	engineer_name	transport_ref
gcss_case		transport_type
loan_order		awb
part_number		transport_company
loan_value		destination
loan_date		origin
order_reason		request_date
return_type		pickup_date
part_status		delivery_date
shipping_wh		
sn_status		
shipto_name		
shipto_address		
shipto_phone		
igso_lmt		
extention_date		
extention_reason		
extention_requestor		
chase_status		
status_code		
last_status_change		
last_info_update		
blocked		

Tabla 5.1: Campos de búsqueda

La lista de campos editables determina aquellos campos de la base de datos que deben poder ser editados desde la interfaz correspondiente de detalles de un componente o de un transporte. A continuación se listan los campos en las tablas 5.2, 5.3 y 5.4, agrupados por las tablas de la base de datos a las que pertenecen.

loan1	**loan2**	**transport**
shipto-address	fsl_part_status	transport_ref
shipto-phone	fsl_transport_incidence	awb
extention_reason	chub_part_status	transport_company
extention_requestor	chub_transport_incidence	destination
igso_lmt	crwb_part_status	origin
blocked	crwb_transport_incidence	request_date
		pickup_date
		delivery_date

Tabla 5.2: Campos editables

adjustment	**comment**	**operation**
adjustment_reason	comment_text	order_type
		operation_order
		period
		operation_date
		operation_value

Tabla 5.3: Campos editables

handling	**rev_receipt**	**blind_receipt**
receipt_date	received_date	zsbr_type
process_date	received_plant	zsbr_rma
fsl	revceived_pn	scanned_event
gidr_ref	receiver_comments	zsbr_reason
gidr_date	received_value	
gidr_issue		

Tabla 5.4: Campos editables

5.5. Migración y carga inicial de datos

No es necesaria ninguna migración de los sistemas actuales a la nueva aplicación, ya que el objetivo es que sea utilizada de manera adicional a los sistemas existentes, como complemento a éstos.

Lo que sí sucede es el cambio en ciertos procesos que se han realizado de manera manual con el uso de herramientas más básicas como hojas de cálculo, que se comenzarán a realizar utilizando la aplicación de este proyecto.

En cuanto a la carga inicial de los datos, se realizará mediante la funcionalidad de la aplicación de Actualización de base de datos a partir de feeds. La condición inicial será la base de datos prácticamente vacía, y el sistema cargará todos los datos necesarios en las tablas correspondientes a partir de los ficheros suministrados.

5.6. Validación de la interfaz del sistema

Para asegurar que el código HTML cumple el estándar especificado en los requisitos, se ha sometido el código al test de validación W3C [14] (ver logo en la figura 5.20).

Figura 5.20: Logo de W3C

A continuación se presenta el resultado de ejecutar el validador sobre el código HTML de la aplicación (ver figura 5.21). Para realizar los tests se ha utilizado la opción *Validate by Direct Input* del validador W3C.

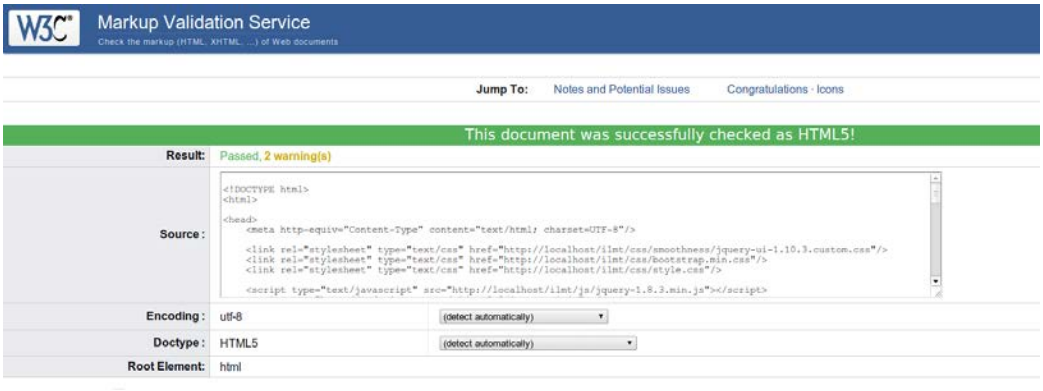

Figura 5.21: Resultado de la validación W3C

Se ha validado por separado el código fuente de cada sección de la aplicación (*Loan search, Transport search, Database update, Adjustments, Engineers, Users*).

Capítulo 6

Gestión del proyecto

El desarrollo del proyecto se ha compuesto de las siguientes fases:

- Análisis del problema: esta fase ha consistido en la investigación y estudio del problema existente, principalmente a través de la comunicación con mi tutora en la empresa.
- Diseño de la solución: en esta fase se ha realizado inicialmente un primer diseño de la solución, teniendo en cuenta los requisitos recogidos durante el análisis del problema. A continuación, una vez comenzada la fase de implementación, se han realizado sucesivas iteraciones sobre el diseño, para adaptarlo a los problemas e imprevistos surgidos durante la implementación, así como a pequeñas modificaciones respecto a los detalles del problema.
- Implementación: durante esta fase se ha escrito el código de la aplicación a partir del diseño creado, realizando un proceso de aprendizaje e investigación de las tecnologías en las que no contaba con gran experiencia.
- Validación: esta fase ha consistido en la instalación, configuración y prueba de la aplicación en un servidor local de la empresa para verificar el correcto funcionamiento de la misma.
- Memoria: en esta fase se ha desarrollado la memoria en la que se documenta con detalle el proceso de creación de la aplicación, desde su inicio hasta su fin.
- Mejoras: esta fase comprende todas las mejoras y correcciones que se han realizado en el código de la aplicación una vez finalizada la

implementación inicial.

Las fases del proyecto se pueden dividir en dos períodos. Desde Febrero hasta Agosto de 2011, se llevó a cabo el desarrollo de la aplicación durante el período de prácticas en la empresa.
A partir de Septiembre de 2011 comenzó un período de gran cambio para mí, con una beca de estudios Erasmus en Finlandia, comenzando un trabajo en el mismo país después de finalizar el año Erasmus que continúa hasta la fecha actual. Todo esto causó la prolongación de la redacción de la memoria del proyecto durante todo este largo período de tiempo y asimismo la voluntad de realizar ciertas correcciones y mejoras pasado un tiempo sustancial desde la implementación de la aplicación.

A continuación se puede ver en la figura 6.1 el diagrama de Gantt, que muestra el progreso de las fases del proyecto a lo largo del tiempo, desde su inicio hasta la fecha actual.

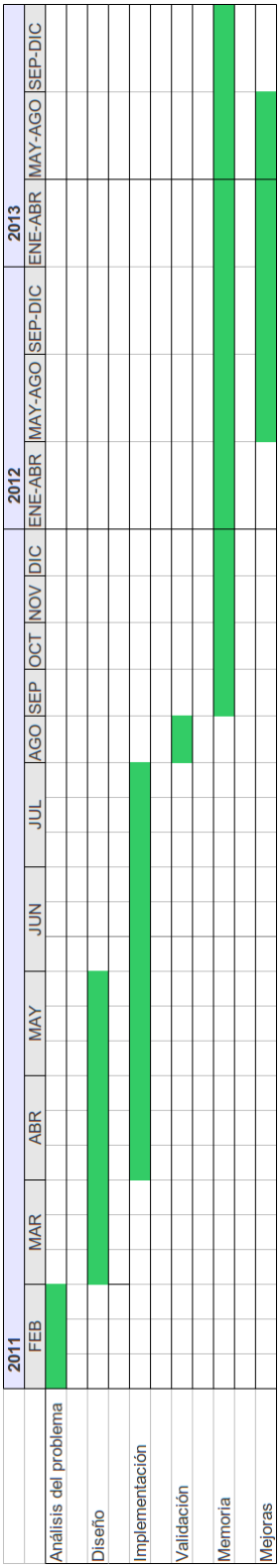

Figura 6.1: Diagrama de Gantt

Capítulo 7

Conclusiones

En este capítulo se recogen conclusiones generales acerca del proyecto y posibles líneas de trabajo futuro.

7.1. Conclusiones generales

Bajo mi punto de vista, este ha sido un proyecto muy interesante, a pesar de haber sido un duro y largo desarrollo, especialmente en la última fase de redacción de la memoria.

Se han cumplido de manera satisfactoria los objetivos propuestos por la empresa. Se ha desarrollado una aplicación que cubre todas las necesidades especificadas y permite mejorar sustancialmente los resultados del trabajo de logística realizado por la empresa.

En la fase inicial, he aprendido muchísimo realizando la aplicación en la empresa, ya que el contexto ha sido básicamente de autoaprendizaje. En el departamento de la empresa, ningún empleado tenía un perfil técnico, lo cual supuso una ventaja y desventaja al mismo tiempo.

La desventaja derivaba de la imposibilidad de contrastar con nadie las múltiples dudas acerca de la aplicación, problemas de diseño, etc.

La ventaja fue una consecuencia del hecho de no poder contar con asesoramiento, pues tuve que desarrollar mi habilidad de autoaprendizaje, viéndome obligado a tomar muchas decisiones de diseño sin tener experiencia alguna. Esto,

por otro lado, es la causa de que la arquitectura de la aplicación tenga puntos mejorables, debido a la falta de un guía para resolver todas las dudas importantes.

La otra gran dificultad a la que me he tenido que enfrentar ha sido la escritura del proyecto. Debido a motivos personales (después del año de prácticas en la empresa durante el que he desarrollado la mayor parte de la aplicación, comencé una estancia de un año de intercambio Erasmus en Finlandia, y posteriormente inicio de un trabajo a jornada completa y permanencia constante en Finlandia desde hace aproximadamente dos años), la escritura de este trabajo y la finalización del proyecto se ha retrasado considerablemente, unos dos años aproximadamente.

Durante estos dos años, y especialmente en este último año en el que he estado trabajando en una empresa de software, específicamente realizando programación web, mi experiencia y conocimientos en el área han aumentado de forma considerable. El problema derivado de esto es que al realizar todas las correcciones y modificaciones a la aplicación durante este último período, he sido consciente de todas las decisiones mejorables que tomé en su día durante la implementación inicial. Ha representado una verdadera dificultad realizar ciertas correcciones y mejoras restringiendo el ámbito de las mismas para no abarcar demasiado.

Inicialmente, una primera versión de la aplicación fue puesta en uso, al finalizar el período de prácticas en la empresa, y ha sido utilizada satisfactoriamente desde entonces. No obstante, una vez finalizado este proyecto, la empresa recibirá la última versión de la aplicación, que incluye múltiples mejoras y correcciones, producidas gracias a los conocimientos adquiridos y trabajo realizado durante los dos últimos años.

7.2. Futuro trabajo

Existen múltiples áreas de la aplicación que podrían ser rediseñadas para mejor mantenibilidad y extensibilidad.

A grandes rasgos, la arquitectura de la aplicación y la estructura de los controladores podría ser rediseñada, aplicando múltiples patrones de diseño, para enfocar la aplicación en una dirección más modular, en la que las clases grandes sean sustituidas por clases más pequeñas y simples conectadas entre sí, de manera que la modularidad de la aplicación mejoraría considerablemente.

Además, al trabajar con clases pequeñas y modulares, se pueden definir tests unitarios para aumentar la seguridad de la aplicación frente a cualquier cambio y aumentar la salud del código, facilitando encontrar cualquier posible fallo.

En la fase final del proyecto, durante las últimas correcciones de la aplicación, se han creado algunos tests para los últimos módulos desarrollados. Este tipo de desarrollo, siguiendo la metodología *TDD* (Test driven development) (ver definición en el glosario) genera como resultado un código más modular, flexible y seguro.

Glosario

Ajuste es el término asignado en el ámbito de la empresa al proceso de finalizar los intentos de recuperación de un componente después de haber sido enviado al cliente o entregado al ingeniero para una reparación. 93

Backend en desarrollo web, hace referencia a la parte del software que se ejecuta en el servidor web. También denominado lado del servidor. 93

Caso es una incidencia a tratar por el departamento de logística con un cliente determinado, relativa a un fallo de hardware en una o varias máquinas (servidores, ordenadores de sobremesa, ordenadores portátiles, etc.). Afecta a uno o múltiples componentes. 93

Componente es un dispositivo de hardware que forma parte del funcionamiento de un ordenador o máquina. 93

Feed es un fichero de datos en un formato específico, actualizado de manera regular, a partir del cual se actualiza la base de datos del sistema. 93

Framework es una estructura de soporte o conjunto de bibliotecas, orientada a la reutilización de componentes con la finalidad de desarrollar aplicaciones software. 93

Frontend en desarrollo web, hace referencia a la parte del software que se ejecuta en el navegador del usuario. También denominado lado del cliente. 93

Generador de código es un tipo de programa que genera un nuevo programa o código. Gracias a este tipo de utilidad, un programador puede escribir código a un nivel superior de abstracción. 93

Hash es una función resumen que toma una entrada y genera un valor distribuido uniformemente en un rango determinado. En este contexto se utiliza como representación compacta de la cadena de entrada, de manera que comparando dos valores hash se puede determinar si los valores de entrada son idénticos o no. 93

Ingeniero es un trabajador de la empresa que realiza la parte física del servicio de soporte técnico que provee la empresa. Se encarga de recoger los componentes nuevos necesarios en los puntos de recogida correspondientes, viajar a la localización de los clientes, y realizar las reparaciones o sustituciones necesarias de componentes. Al finalizar los trabajos, debe devolver los componentes nuevos o sustituidos a la empresa. 93

Inyección de dependencias es un patrón de diseño software orientado a objetos cuyo objetivo principal es independizar los componentes de software y obtener un diseño más modular y flexible. Esto se consigue suministrando a un objeto sus dependencias (módulos de software de los que el objeto depende) a través de parámetros, en lugar de ser el propio objeto el que las defina. 93

Librería es un conjunto de implementaciones en un lenguaje de programación, que provee utilidad a otros programas a través de una interfaz. 93

ORM (Object Relational Mapping) es una técnica de programación para convertir datos entre dos sistemas incompatibles utilizada en lenguajes orientados a objetos. En el contexto de una aplicación MVC, se refiere a la conversión de los datos entre un modelo definido mediante orientación a objetos en el lenguaje de programación correspondiente y un modelo relacional de base de datos. 93

Préstamo es el término estipulado en el ámbito de la empresa al proceso de entregar un componente a un ingeniero o al cliente final con el fin de realizar una reparación o sustitución. Se denomina préstamo ya que el proceso persigue obtener un componente de vuelta a la empresa, ya sea el componente defectuoso sustituido, o el componente nuevo en caso de no haber sido necesaria su utilización. 93

TDD (Test Driven Development) es un proceso de desarrollo de software basado en la repetición de un ciclo corto de desarrollo en el que el

programador escribe inicialmente un test unitario para una pequeña funcionalidad a implementar. Este test inicialmente falla, ya que la funcionalidad aún no está implementada. A continuación, el programador implementa la funcionalidad hasta que el resultado del test es satisfactorio. 10, 93

Bibliografía

[1] The Apache HTTP Server Project. `http://httpd.apache.org/`. Accessed: 2013-11-30.

[2] Twitter Bootstrap. `http://getbootstrap.com/2.3.2/`. Accessed: 2013-11-30.

[3] CakePHP. `http://cakephp.org/`. Accessed: 2013-11-30.

[4] Codeigniter. `http://ellislab.com/codeigniter`. Accessed: 2013-11-30.

[5] David Flanagan. *JavaScript: the definitive guide*. O'reilly, 2011.

[6] Javascript | Mozilla Developer Network. `https://developer.mozilla.org/en-US/docs/Web/JavaScript`. Accessed: 2013-11-30.

[7] jQuery. `http://jquery.com/`. Accessed: 2013-11-30.

[8] Helmut Kopka and Patrick W Daly. *Guide to LATEX*. Pearson Education, 2003.

[9] Oracle MySql. `http://www.mysql.com/`. Accessed: 2013-11-30.

[10] PHP: Hypertext Preprocessor. `http://www.php.net/`. Accessed: 2013-11-30.

[11] Symfony. `http://symfony.com/`. Accessed: 2013-11-30.

[12] Kevin Tatroe, Peter MacIntyre, and Rasmus Lerdorf. *Programming Php*. O'Reilly Media, Inc., 2013.

[13] TIOBE Software: Tiobe Index. `http://www.tiobe.com/index.php/tiobe_index`. Accessed: 2013-11-30.

[14] The W3C Markup Validation Service. http://validator.w3.org/. Accessed: 2013-11-30.

[15] Zend Framework. http://framework.zend.com/. Accessed: 2013-11-30.